The Subgoal Structure as a Cognitive Control Mechanism in a Human-Computer Interaction Framework

Hee Sen Jong
University of Michigan

Research and Advanced Concepts Office
Michael Drillings, Acting Director

March 1996

19960815 147

United States Army
Research Institute for the Behavioral and Social Sciences

DTIC QUALITY INSPECTED 1

U.S. ARMY RESEARCH INSTITUTE
FOR THE BEHAVIORAL AND SOCIAL SCIENCES

**A Field Operating Agency Under the Jurisdiction
of the Deputy Chief of Staff for Personnel**

EDGAR M. JOHNSON
Director

Research accomplished under contract
for the Department of the Army

University of Michigan

REPORT DOCUMENTATION PAGE

1. REPORT DATE 1996, March	2. REPORT TYPE Interim	3. DATES COVERED (from. . . to) December 1988-December 1993	
4. TITLE AND SUBTITLE The Subgoal Structure as a Cognitive Control Mechanism in a Human-Computer Interaction Framework		5a. CONTRACT OR GRANT NUMBER MDA903-89-K-0025	
		5b. PROGRAM ELEMENT NUMBER 0601102A	
6. AUTHOR(S) Hee Sen Jong (University of Michigan)		5c. PROJECT NUMBER B74F	
		5d. TASK NUMBER 1901	
		5e. WORK UNIT NUMBER C19	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) University of Michigan Division of Research and Development 475 East Jefferson, Room 1318 Ann Arbor, MI 48109-1248		8. PERFORMING ORGANIZATION REPORT NUMBER	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) U.S. Army Research Institute for the Behavioral and Social Sciences ATTN: PERI-BR 5001 Eisenhower Avenue Alexandria, VA 22333-5600		10. MONITOR ACRONYM ARI	
		11. MONITOR REPORT NUMBER Research Note 96-58	

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES
COR: Michael Drillings

14. ABSTRACT *(Maximum 200 words)*:
Human-Computer Interaction (HCI) research has gained prominence due to the need to make computers easier to learn and use. This research (1) develops an HCI framework to structure and review HCI models, (2) develops a subgoal theory that investigates some pieces missing from current models, and (3) tests the subgoal theory.

15. SUBJECT TERMS
Human-Computer Interaction   Subgoal theory   Task   Formulas

SECURITY CLASSIFICATION OF			19. LIMITATION OF ABSTRACT	20. NUMBER OF PAGES	21. RESPONSIBLE PERSON (Name and Telephone Number)
16. REPORT Unclassified	17. ABSTRACT Unclassified	18. THIS PAGE Unclassified	Unlimited	205	

i

ACKNOWLEDGEMENTS

I am most grateful to Professor Judith Olson, who not only helped shape this dissertation, but also helped shape my research interest and skill. Her support, care, encouragement, stimulating insights, and generosity with her time have greatly enriched my graduate studies experience.

I would like to thank my committee members for making the dissertation research challenging yet fun. Professor Gary Olson, thank you for helping me to be precise in stating the theories. Professor Marlys Lipe, thank you for taking much time and care to help polish the data analyses and the presentation in this dissertation. Professor Jeffrey Kottemann, thank you for providing perspectives from outside the Human-Computer Interaction field.

I wish also to express my gratitude to the National University of Singapore (NUS) and to Professor Judith Olson for the financial support for my graduate studies. This dissertation is also supported in part through a grant from the Army Research Institute (026147) to Judith Olson. I also like to thank Professor Yuen Chung Kwong, Head of the Department of Information Systems and Computer Science at NUS, for his encouragement over the years.

Finally, this dissertation can never be completed without the loving support and prayers from friends, family members, and my wife, Cher Leng. I thank my fellowship group for providing prayer and spiritual support. I thank my mum for her sacrificial love and daily prayer. I thank God for blessing me with Cher Leng, whose love, prayer, and inspiration have made everything seems so worthwhile.

TABLE OF CONTENTS

CHAPTER

1 THE RESEARCH PROBLEM

2 A HUMAN-COMPUTER INTERACTION COGNITIVE MODELING FRAMEWORK

3 THE FORMULA EDITOR EXPERIMENT

LIST OF TABLES

LIST OF FIGURES

LIST OF APPENDICES

Appendix

ABSTRACT

THE SUBGOAL STRUCTURE AS A COGNITIVE CONTROL MECHANISM IN A HUMAN-COMPUTER INTERACTION FRAMEWORK

By
Hee-Sen Jong

Chairperson: Judith S. Olson

Human-Computer Interaction (HCI) research has gained prominence due to the need to make computers easier to learn and use. This research (a) develops a HCI framework to structure and review HCI models, (b) develops a subgoal theory that investigates some pieces missing from current models, and (c) tests the subgoal theory.

This HCI framework combines the Action Theory [Norman, 1986] and the SRK Framework [Rasmussen, 1976] to describe the seven stages of HCI activities and the underlying cognitive processes. These activities are classified as knowledge-based, rule-based, or skill-based behaviors, and they are associated with different cognitive controls, stimuli, and errors. This framework clarifies differences in behavior, states hypotheses for investigation, and structures HCI models. A review of the current models reveals that they seldom address knowledge-based behavior.

This research models knowledge-based HCI activities by investigating the subgoal structure as a cognitive control mechanism to overcome bad interfaces. Users employ extra subgoals both when the task and interface operate on different objects, and when they require a different action

sequence for task completion. The bottlenecks in HCI arise from the planning, monitoring, and validating of these extra subgoals.

This research concludes with two studies to demonstrate the subgoal theory's explanatory power. In the formula editor study, subjects used a linear editor and a semantic editor to key in formulas. The linear editor operates on character strings and the semantic editor operates on formula structures. Subjects using the linear editor needed many extra subgoals to translate the mismatched structures, resulting in longer keystroke time, extra keystrokes, and higher error rate.

In the Lotus menu traversal experiment, subjects saw one of two menu organizations and received one of two instruction formats to execute simple spreadsheet tasks. The original Lotus menu was redesigned to have a consistent structure. Subjects using this new menu could form consistent subgoal structures to explore the menu. Task instruction formats were also manipulated to either match or mismatch the redesigned menu structure. Subjects given the new menu with the matching instructions had the best performance as they did not need extra subgoals to buffer the out-of-sequence task actions.

CHAPTER 1

THE RESEARCH PROBLEM

The objective of this research is to further our understanding of the Cognitive Modeling approach in Human-Computer Interaction (HCI) research. Specifically, this research (1) develops a conceptual HCI framework that overlays a cognitive architecture on the stages of HCI activities and uses the framework to point out places ripe for further research in current HCI cognitive models; (2) points out the roles of subgoal formation to address some missing pieces in the above framework and; (3) demonstrates empirically how the subgoal theory explains and predicts some bottlenecks when users interact with computer applications.

The following sections in this chapter discuss the background, the motivation, the approach, and the objectives of this research. This chapter concludes with an outline of the chapters in this research.

1.1   Background and Motivation

The motivation of this research springs from the need to improve the cognitive modeling approach as HCI issues gain importance. This section explains why HCI research is important, and in particular the cognitive modeling approach. It points out why it is impractical to have a comprehensive cognitive model of performance and why there is a need to develop a conceptual HCI framework.

1.1.1 The Importance of HCI Research

The objective of human-computer interaction (HCI) research is to produce theories and tools to design better interfaces for computer applications. HCI research has gained importance and prominence as computer usage becomes essential to compete in the marketplace; and as computer applications become more complex yet widespread. The computer users are no longer only computer professionals who are motivated to adapt to complicated interfaces, but are now mostly discretionary end users who are often frustrated by incomprehensible and intimidating systems [Bertino, 1985]. There is an urgent need to make computer applications easier to use for these discretionary users to realize the full impact and payoff of computer powers.

A group of computer industry leaders echo this concern and stress the importance of HCI research in a recent CACM[1] article [Straub & Wetherbe, 1989]. In that article, the authors interviewed eight Presidents or Chairmen of the largest information system (IS) consulting companies and three prominent business school IS professors in the United States. The interview results identify improvement in human-computer interface as the aspect of technology that has the greatest organizational impact and payoff in the 1990s. The IS leaders cite underutilization as the reason for the push for improvement in HCI design. They state that existing interfaces have not reached an acceptable level of usability thus causing underutilization since non-computer-literate users find systems too difficult to use. They believe that if the current technology can be fully utilized, the payoff is enormous in terms of productivity gained and competitive edge.

This end users' demand for HCI research coupled with the technology push of rapid hardware and software innovation has led to recognition and acceptance of HCI

---

[1]Communications of the Association of Computing Machinery.

research. This in turn led to increased research activity, as well as increased expenditure on the user interface in product development [Lewis, 1990]. The portion of code devoted to the interface has now risen to 70% in an average end-user application [Dertouzos, 1990]. There is also an increasing number of related conferences and publications. Although the practical importance of HCI research is now well established, the proper role of research in the development of the technology and the kind of research that is appropriate remain in question [Lewis, 1990]. However, there is an emerging trend of HCI research that emphasizes the development of theory-based HCI performance models. This trend views the central goal of HCI research as extending psychological theory to account for differences among interfaces. The claim is that adequate theory would permit designers to predict the performance of designs without resorting to empirical comparisons.

1.1.2 The Push Towards Cognitive Modeling

The HCI field is young but has made good progress. The research paradigm has progressed from empirical comparison of implementations, to the development of design guidelines, and recently to the development of HCI design theories using a cognitive modeling approach.

Early HCI research focused on experimental comparisons of the merits of implementations, following the human factor research tradition. The wealth of this research produced a body of literature on the performance parameters for interacting with computer devices and a collection of guidelines for designing interfaces. This body of knowledge serves to show the usefulness of testing with real users, but this empirical approach has its limits.

Making complex systems more usable requires behavioral research, but standard experimental methodologies are not sufficient. The classical experimental-control

comparison can be useful in deciding whether to add a particular feature to a system but it produces little information about the underlying processes from which one can generalize [Landauer, 1988]. Also, it is too costly and time consuming. It is simply not feasible to perform formal experiments for resolving all issues that are likely to arise within the context of a particular design problem and its associated development cycle [Barnard, 1987]. There is a need for theoretical models that can provide answers to common interface design problems without always resorting to empirical test [Reisner, 1987].

The theoretical background for HCI research is cognitive psychology. In attempting to predict and understand users' behavior at the human computer interface, we need to study higher cognitive processes, e.g., memory, perception, learning, problem solving, etc., to understand users from a cognitive point of view. To become skilled in using a computer application, it is the user's conceptual and cognitive skills, not the perceptual motor skill, that must be automated [Wilson,Barnard,Green, & MacLean, 1988]. We need to understand what users understand and how they understand it and the underlying cognitive processes of their behaviors. Norman calls the application of findings and theories in cognitive psychology to HCI research *cognitive engineering* [Norman, 1987]. From theories in cognitive psychology, researchers derive HCI cognitive models that attempt to explain the cognitive processes involved in HCI. Understanding these processes will help us design better interfaces without resorting to empirical comparison.

1.1.3 The Benefits of HCI Cognitive Models

There are many benefits of using HCI cognitive models as design tools besides the economy of not having to resort to empirical evaluation whenever a new feature is added or a new piece of software is developed. This power of prediction without resorting to empirical evaluation comes from the theory-based explanations furnished by

the models. The theory-based models help designers generalize from one design situation to another. The theory-based models allow prediction of some important parameters, e.g., performance time, error rate, learning time, etc., even before the product is developed, by plugging the new task and interface parameters into the models. Since cognitive models can be used to evaluate individual portions of a new system, evaluation need not wait until the product is fully developed. This makes early and partial evaluation possible. Consequently, using cognitive models as design tools reduces the need to develop ad hoc design techniques, hastens the process of application development, and saves time and money by reducing empirical testing of products.

There are other less obvious but important benefits when cognitive models are used as design tools. The process of using the model helps a designer to be more thorough because the model forces the designer to look the most relevant issues. The model will help highlight the potential complexity of the new product and thus will sensitize designers to the complexity users face. The models help designers focus on the most important design issues. This helps narrow the design space and suggests improvements to the current design. Having a prescribed method of applying the model also makes evaluation simpler and more directed.

Cognitive models are not only useful interface design tools, they also help in creating more effective training material [Elkerton & Palmiter, 1991]. Cognitive models enable a deeper understanding of the knowledge required for the task. Making user knowledge explicit in a model helps in designing training materials to take advantage of learning principles such as transfer of knowledge, interference, etc. Making user knowledge explicit also will help predict where users will face difficulties and commit errors. The training material can then be designed specifically to address how to overcome these difficulties, how to avoid these potential errors, and how to recover from such errors if committed.

1.1.4 The Limited Success of HCI Cognitive Models

Despite these potential benefits, current HCI cognitive models are not widely used. Practitioners do not use the models to design real life applications [Bellotti, 1987]. HCI models have primarily been used and tested in a laboratory context where the task environments are somewhat restrictive. The models are cumbersome and unwieldy to apply to life-size applications. The critics of cognitive modeling approach criticize the modeling methodology as unrealistic [Carroll & Campbell, 1986]. The models have at most been used by researchers to evaluate a small portion of an existing application as examples of applicability rather than in service of a real design issue.

One big inherent problem of HCI cognitive models is that they are fragmentary and do not address all aspects of the interaction. Two factors hinder the development of an all comprehensive HCI cognitive model. First, a comprehensive model will be too complicated to put to practical use. Second, the underlying cognitive theories are themselves fragmented and restrictive in scope [Barnard, 1987; Carroll, 1990]. Theories in cognitive psychology are typically partial theories focused on a specific component of mental life — for example, speech, perception, short-term memory, or problem solving[2]. Carroll commented that the key problem of HCI cognitive models is that both the concepts and the methods in basic psychology have been specialized for simple and abstract situations [Carroll, 1990]. However, in the rich task environments of HCI, the processes of problem solving, memory retrieval, perception, recognition, and sensory-motor control are equally important and coordinate among themselves. These interweaving processes seem to require an integrated theoretical approach to address all the cognitive resources involved. Since we may not be ready for a unifying HCI theory

---

[2]Allport discussed the fragmentary and paradigm-bound nature of cognitive theory [Allport ,1980].

because it may be too complex and there is no underlying unifying cognitive theory of human performance, the next best thing is to develop an HCI framework to unify and structure these fragmented HCI theories.

1.1.5 The Call for an HCI Framework

There is a need for a theoretical HCI framework for understanding the whole HCI process, for understanding what occurs at the interface, and for contributing towards the development of techniques where the cognitive issues are addressed [Booth, 1989; Reisner, 1987]. An HCI framework helps to classify HCI issues as it provides a language in which to talk about and clarify different types of behavior as a basis for system design. Classification is an intrinsic part of the development of a new science [Fleishman & Quaintance, 1984] and it leads to the stating of hypotheses for further investigation.

To be useful and practical, the HCI framework needs to identify the activities involved when using a computer application, and the underlying cognitive processes involved in these activities. This framework will then provide a means for highlighting different theories from related fields, like cognitive psychology, artificial intelligence, linguistics, etc., that are relevant to the different aspects of the framework. This will help researchers adapt these theories to HCI cognitive models.

The HCI framework also provides a means to review and structure the HCI literature. The framework can identify how current HCI cognitive models fit into the framework and areas that need further research. By knowing the particular aspects of the framework current HCI models address, it helps interface designers identify models that are appropriate for different design situations. This is necessary as models are best applied locally to small aspects of the design where they fit the criteria of the design decisions [Moran, 1986].

The framework also helps identify aspects of cognition that current HCI cognitive models do not address. In this research, the HCI framework helps identify the role of subgoals for further research. This illustrates the generative, integrative power of constructing such a framework.

1.2   Research Objectives and Approach

The objectives and approach of this dissertation are three fold: (1) to develop an integrative HCI framework to further our understanding of the HCI cognitive modeling approach and to use the framework to review and structure the HCI literature; (2) to investigate a missing piece of the framework not addressed by current models by modeling the roles of subgoal formulation for cognitive control; (3) to validate the model empirically using common applications. The HCI framework will be derived from theories in the HCI, the cognitive science, and the human factors literature. We then use the framework to review the current HCI models and identify the roles of subgoals as one aspect not modeled. A theory of the role of subgoals as a bridge between the task space and the system space is developed, and followed finally by empirical demonstration of behavior consistent with the subgoal theory.

Develop a Framework

This research will develop an integrative HCI framework that encompasses the various cognitive processes and activities involved when interacting with a computer application. The framework will be derived from theories in cognitive science, HCI, and industrial control systems. This framework will include the stages of activities, the types of cognitive control used, the types of possible errors, and the types of stimuli processed during interactions. The framework will then be used to review the HCI modeling literature. The review of the literature points to the need for research to investigate the use of subgoals in HCI behavior.

Investigate the Subgoal Structure as a Cognitive Control Mechanism

Mapping our psychological goals to the physical variables and controls of the task has been identified in the literature as a bottleneck in HCI [Moran, 1981; Norman, 1987; Young, 1981]. Yet, there have been few models that explicitly address this important process. We will discuss the subgoal structure as a cognitive control mechanism that bridges the task space (the psychological goals) and the system space (the physical variables and operations). We will develop a theory that points out that a bad interface causes users to create extraneous subgoals that do not contribute directly to the task goals but only help users cope with the bad interface. We identify two situations where the interfaces are bad: (a) when the task space and system space operate on different object structures; and (b) when the system space does not allow users to carry out actions in the sequences conceived in the task space. The acquisition, monitoring, and checking of these extra subgoals require a long time and often exceed memory capacity, causing erroneous behaviors.

Test the Subgoal Theory Empirically

We will use two common applications to validate our theory. In the first experiment, we show how users create extra subgoals causing poor performance to bridge the task-system space when using a linear text editor to enter formulas. The formulas in the task space have a hierarchical structure but the linear editor in the system space operates on a linear character string. A simple redesign of a formula editor shows how extra subgoals contributing to memory load are drastically reduced when using the new editor. Also, by allowing a direct mapping between the formula structure and the structure of the editor, the need for planning and checking is greatly reduced.

In the second experiment, a simple redesign of the menu of an existing spreadsheet package, Lotus 123, shows that an interface that allows users to execute task

actions in the sequence as conceived in the task space leads to superior performance by eliminating extra subgoals. Consistent mapping between how the users think of the task and the menu structure improves learning, performance, and reduces error rates. Consistent mapping reduces the need for extra subgoals to restructure users' conception of the task.

1.3   Outline of the Chapters

The rest of this thesis will:

(i)   develop the HCI framework,

(ii)   review the relevant HCI literature on cognitive modeling,

(iii)   describe the subgoal theory, and

(iv)   describe the experiments and results to verify the subgoal theory

Chapter 2 develops the HCI framework from theories in the cognitive science, HCI, and human factors literature. The framework will be used to structure and review the current HCI models. The second part of Chapter 2 develops the roles of subgoal formulation as a cognitive control mechanism to account for non-skilled behavior, the missing link in many popular HCI models. This subgoal theory will set up the grounds and rationale for the experiments described in the next two chapters.

Chapter 3 describes the rationale, the hypotheses, the methodology, and the results for the laboratory study on the effect of using different formula editors on the process of mapping task goals to subgoals. By allowing a direct mapping between the formula structure and the structure that the editor manipulates, the need for extra subgoals, planning, and checking are greatly reduced.

Chapter 4 describes the rationale, the hypotheses, the methodology, and the results for the laboratory study on the effect of different menu and instruction structures on the process of mapping task goals to subgoals. By allowing a direct mapping between

the form of instruction and the menu structure, the need for extra subgoals again is reduced leading to improved learning and reduced error rates.

Finally, Chapter 5 suggests potential contributions and implications of this research, and outlines a design methodology for applying the subgoal theory.

CHAPTER 2

A HUMAN-COMPUTER INTERACTION

COGNITIVE MODELING FRAMEWORK

This chapter begins with the development of a human-computer interaction (HCI) framework that imposes a cognitive architecture on the different stages of activities when users interact with computer applications. This framework will not only help us review and structure the HCI literature and point us to areas that need future research, it will also help HCI practitioners choose the appropriate HCI models in their design process by matching the current design criteria to the models' strength and coverage.

The chapter concludes with the development of a subgoal theory about the subgoal structure as a cognitive control mechanism in the HCI framework. In our framework, users plan their task by decomposing the task goal into a series of action subgoals that are within the users' performance limitations. This link is not well addressed by the current HCI cognitive models and can be shown to be the bottleneck for some tasks even for expert users. The subgoal structure can also be used to address more problem solving behavior, which is not modeled by current HCI models like GOMS [Card,Moran, & Newell, 1983] or TAG [Payne & Green, 1986]. This theory will set the grounds and premises for the experiments described in Chapter Three and Chapter Four.

2.1   The HCI Framework

The HCI framework evolves around the task as the task is the focus and objective of using any computer tools. The purpose of the user interface is to enable people to use

12

the computer to help them perform some tasks [Johnson, 1985]. The HCI framework will have to address the various stages of physical and cognitive activities involved when performing tasks through the interface.

A task has been defined as: "The achievement of a set of goals while maintaining a set of constraints" [Filkes, 1982]. When interacting with a computer, the user evaluates the situation and decides the task goals to accomplish. The task goals are derived from the task domain. In some cases the user knows exactly what the task goals are and the procedures to accomplish them. In such cases we can say that the task is routine. In other cases the user may be clear about the task goals but uncertain about the procedures to choose, and these cases can be classified as problem-solving situations. Finally, there may be cases in which the task goal is not altogether well-defined and it is the function of the user to create and identify a satisfactory result. These situations can be described as creative. Our framework will deal only with situations where the task goals are well defined but the users may or may not know how to accomplish the task goals through the interface.

For each task goal, the user initiates a plan to accomplish it with a series of subgoals. Depending on how adequately the interface supports the execution of the task goal and the user's knowledge of how to execute the task goal, the subgoal structure may be complex or simple, or even unnecessary if there is a simple method to achieve the task goal. The subgoals are chosen so that they can be performed within the set of cognitive constraints. The cognitive constraints arise due to limitation of the user's cognitive resources like working memory, the knowledge of the methods, etc. After the subgoals are formulated, the user then chooses the procedures and actions to accomplish each subgoal and perceives the feedback from the system and evaluates whether the subgoals have been accomplished.

This cycle of the stages of activities—plan, execute and evaluate—will be the core structure of our HCI framework. On top of these stages of activities, we will impose a cognitive architecture to specify the underlying cognitive activities and cognitive resources required at each of these stages.

2.1.2 The Stages of Activities

We will adapt Norman's Action Theory [Norman, 1986] to model the stages of activities when interacting with a device. This qualitative theory describes seven stages of activities that elaborates on the plan-execute-evaluate cycle discussed above. One part of the theory concerns the production of an action; the other part involves the feedback loop. The seven stages of activities are: forming the goal, forming the intention, specifying an action, executing the action, perceiving the system state, interpreting the system state, and evaluating the outcome. Figure 2.1 depicts the Action Theory.

Figure 2.1 Norman's Action Theory
(Adapted from [Norman 1986])
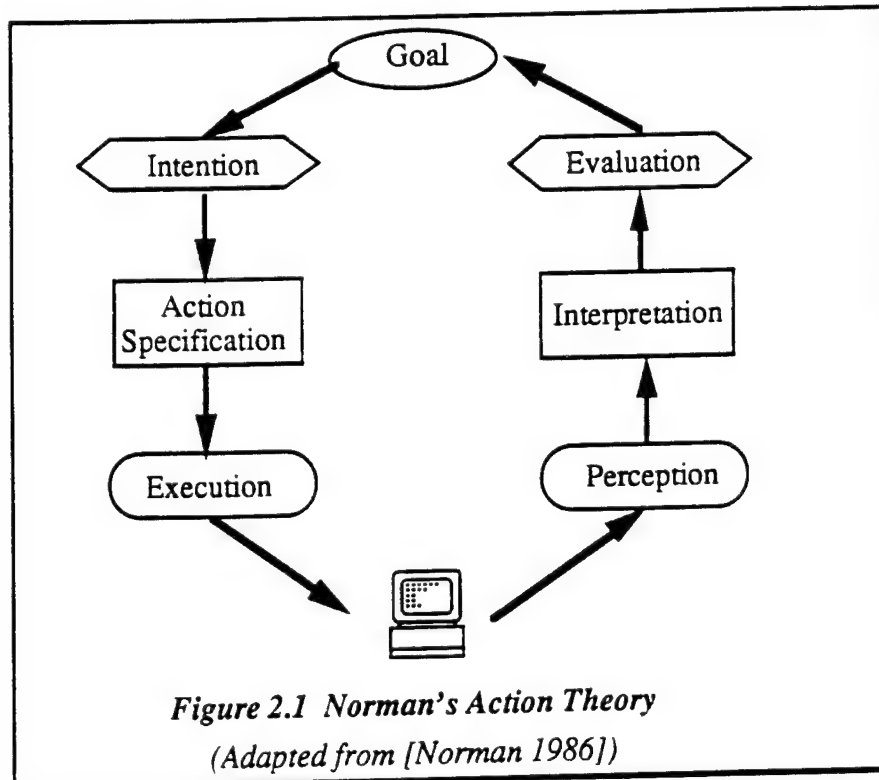
In the Action Theory, the *Goal* is very general. To use Norman's example, the goal can be as general as improving the appearance of a letter [Norman 1986, page 44]. From this general goal, the user generates a few intentions, for example, center the headings, delete a blank line, etc. For each intention, the user needs to specify some actions to achieve the intention. After the user executes the actions by physically typing or moving the mouse, the user needs to perceive, interpret and evaluate the feedback from the interface and sees whether the intention has been met. Further intentions may be generated in the process.

Task Goal and Action Subgoals

For our framework, we modify the Action Theory to make the Goal more specific. We narrow the general Goal into the more specific Task Goals that are derived from the task environment cues, e.g., a marked up manuscript for editing will have clear individual task goals corresponding to the changes to be made. Using Norman's example, our task goals will be the more specific *delete the line, center the heading,* etc. In our framework, we want to distinguish goals that are generated in the task space and action subgoals that are generated as a result of having to use the interface. Task goals are generated from the task environment but subgoals are planned by users to achieve the task goal. The action subgoals are generated in and constrained by the system space, e.g., what operators are available, what methods the user knows, what objects the system manipulates, etc. The subgoals are a bridge between the task space and the system space. To help clarify the distinctions between goals and subgoals in our framework, we can think of goals as couched in the language of the task space whereas the subgoals are couched in the language of the system space. For example, for the task goal of deleting a blank line, the subgoals will be moving the *cursor* to the line and issuing the *command* through the *menu*. We will thus modify the Action Theory's Intention stage to be the *Subgoals*

Formulation stage and modify the Goals stage to be the more specific *Task Goal Formulation* stage.

Our framework and theory will not deal with situations where the users do not know what the task goals are, previously identified as a creative process. This is beyond the scope of this framework[1]. Thus, our user will know that in order to make the letter look nice, he needs to center the heading and delete the blank line. Any practical HCI model will not handle the case where the user does not know the task goals or the desired results. The user may or may not know the method to achieve that task goal using the interface but the user has to know the desired results.

This separation of the task goal from the subgoals of interaction is very important for both designing and evaluating interfaces. Some tasks take a long time to execute because the task goals are very difficult to formulate, e.g., creative design; whereas some tasks take a long time to execute because the interface does not support the direct execution of the task goals, e.g., a textual command-based drawing application. We cannot aid the user by changing the interface in the former situation but we should certainly improve the latter interface. We want to design interfaces that are transparent, such that users can think in terms of concepts in the task space rather than those in the system space to the greatest extent possible.

Method Specification

In our reformulation of the Action Theory, users will have a corresponding method or procedure that specifies the actions to accomplish each subgoal created. We will change Norman's Action specification stage to the Method specification stage. If the user does not have a method for a particular subgoal, the subgoal will have to be further

---

[1]Future research could possibly extend in this direction.

refined until a corresponding method is known. For example, if the user knows the command for deleting a blank line, the subgoal will be to issue the command; if the user does not know the command, the subgoals will be to invoke the menu system, to explore the menu, and to look for the synonym of *delete* and the synonym of *line*. Figure 2.2 shows the slightly different stages of activities in our HCI framework.

Although the Action Theory went on further to explain the gulf of execution and the gulf of evaluation causing a device to be difficult to use, the theory does not address the cognitive mechanisms that creates these gulfs. The Action theory by itself is not explicit about the kinds of cognitive activities, the types of possible errors, and the types of cognitive controls present at the different stages. What we need to complete the HCI framework is to overlay a cognitive architecture on the stages of activities that explicates the cognitive processes underlying these activities. This complete framework can then be used to link HCI activities with human performance theories in the psychology and human factors literature.

2.1.3 Rasmussen's SRK Framework

Our framework will include a cognitive architecture by adopting the Skill, Rule, and Knowledge-based (SRK) classification of human operators' behavior in industrial control systems [Rasmussen, 1976, 1980]. Rasmussen developed the SRK framework to model human operator information-processing abilities and limitations in complex environments, e.g., power plants, chemical plants and aircraft cockpits. (For an overview of the SRK framework, see Rasmussen [1986]). The framework has been embraced by designers, engineers and regulators as providing understandable, usable, and extendable concepts for classifying the behavior of humans in their interaction with the world [Goodstein, Andersen, & Losen, 1986].

Figure 2.2   The Stages of Activities in the HCI Framework
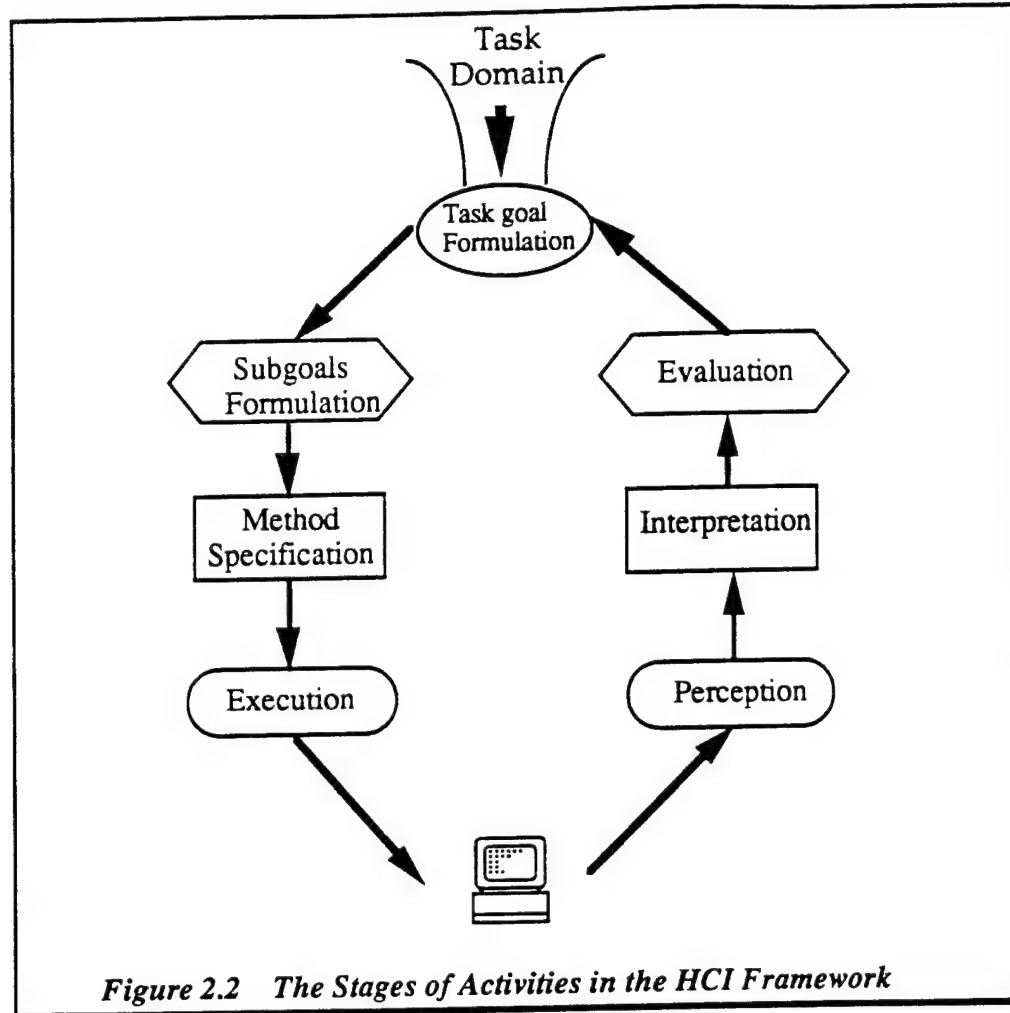
In the SRK framework, complex process operators' behaviors are classified into three categories, skilled-based, rule-based, and knowledge-based. These three categories are the basis of *cognitive control* for the operator's behavior. These three levels of behavior associate with them different types of *stimuli* processed from the environment, and different types of human *errors* during performance.

These three levels of behavior correspond to different levels of familiarity with the task and thus require qualitatively very different types of cognitive control. We can think of the types of cognitive control as the different information processing strategies or modes. At the **knowledge-based level**, the task situation is novel or the actions are too complicated to be controlled by simple retrieval of action rules. At this level, cognitive

control must be planned on-line using conscious analytical processes and previous knowledge. Errors at this level arise from resource limitations like working memory overload, or incomplete or incorrect knowledge. At this level, the stimuli from the environment are evaluated as *symbols* for causal reasoning. At the **rule-based level**, the task is familiar and the behavior is controlled by stored rules. Errors at this level arise from *mistakes* and are related to misclassification of situations leading to the application of the wrong rule, or are related to the incorrect recall of procedures. The stimuli from the environment processed is perceived as *signs* that triggers programmed response at this level. At the **skill-based level**, behavior is controlled by stored sensory-motor programs represented as analogue structures in a time-space domain. Errors at this level arise from *slips* and are related to force, space, or time co-ordination. At this level, the stimuli from the environment processed are perceived as *signals*, as mere feedback to the sensory-motor programs [Rasmussen, 1986].

Similar tripartite classification of human behavior can be found in other prominent research thus giving it validity. Fitts [Fitts & Posner, 1967] proposed three stages of skill learning: cognitive, associative, and autonomous. The same three stages can be found in the cognitive model ACT* [Neves & Anderson, 1981] where the three stages are: encoding, proceduralization, and composition. Also, Card et al. [1983] describe the evolution of cognitive skill as a passage from problem solving behavior, with its emphasis on a problem space, to a more perceptual-motor based cognitive skill. We can therefore graft this proven behavior classification into our HCI framework.

We will superimpose the SRK framework on Norman's Action theory as there are many similarities among activities in HCI and process control. Both require a high level of cognitive skills rather than physical skills alone. We note that Norman's Action Theory is similar to Rasmussen's eight stages of decision making for operators in industrial process plants: activation, observation, identification, evaluation, goal

selection, procedure selection and activation [Rasmussen, 1980]. The difference between the two is in where the cycle of activities starts. In HCI, it starts with the user forming a task goal and then planning the procedures to carry it out. In process control, it starts with the operator reacting to the signals from the displays and then initiating a routine to remedy the situation. In HCI, the user has more control of the interaction and pace as he initiates the interaction; whereas in process control, it is more event driven; the operator monitors the signal from the gauges, interprets them, and reacts if necessary. However, when users and operators are continuously engaged in the feedback, plan, and execute cycle, as in real-life interaction with the devices, the HCI activities and process control activities are identical. We will thus superimpose the SRK framework on Norman's Action theory to make our HCI framework complete with descriptions of both the stages of activities and the underlying cognitive control and processes.

Figure 2.3 shows how we can superimpose the SRK framework on the modified Norman's Action theory. The diagram is stratified into three regions where each region corresponds to one level of cognitive control and the associated type of error and stimulus. The top one-third are knowledge-based activities, the middle one-third are rule-based activities, and the bottom one-third are skill-based activities.

In the next three sections, we discuss how the concepts and ideas from the Rasmussen's SRK framework (adapted from Rasmussen [1986]) can be applied to interpret HCI activities. We find that the explanatory power of the SRK framework can be extended easily to the domain of HCI. We will borrow heavily the ideas from the SRK framework but supplement the discussion with examples specific to the HCI domain.

Figure 2.3   *The HCI Cognitive Framework*
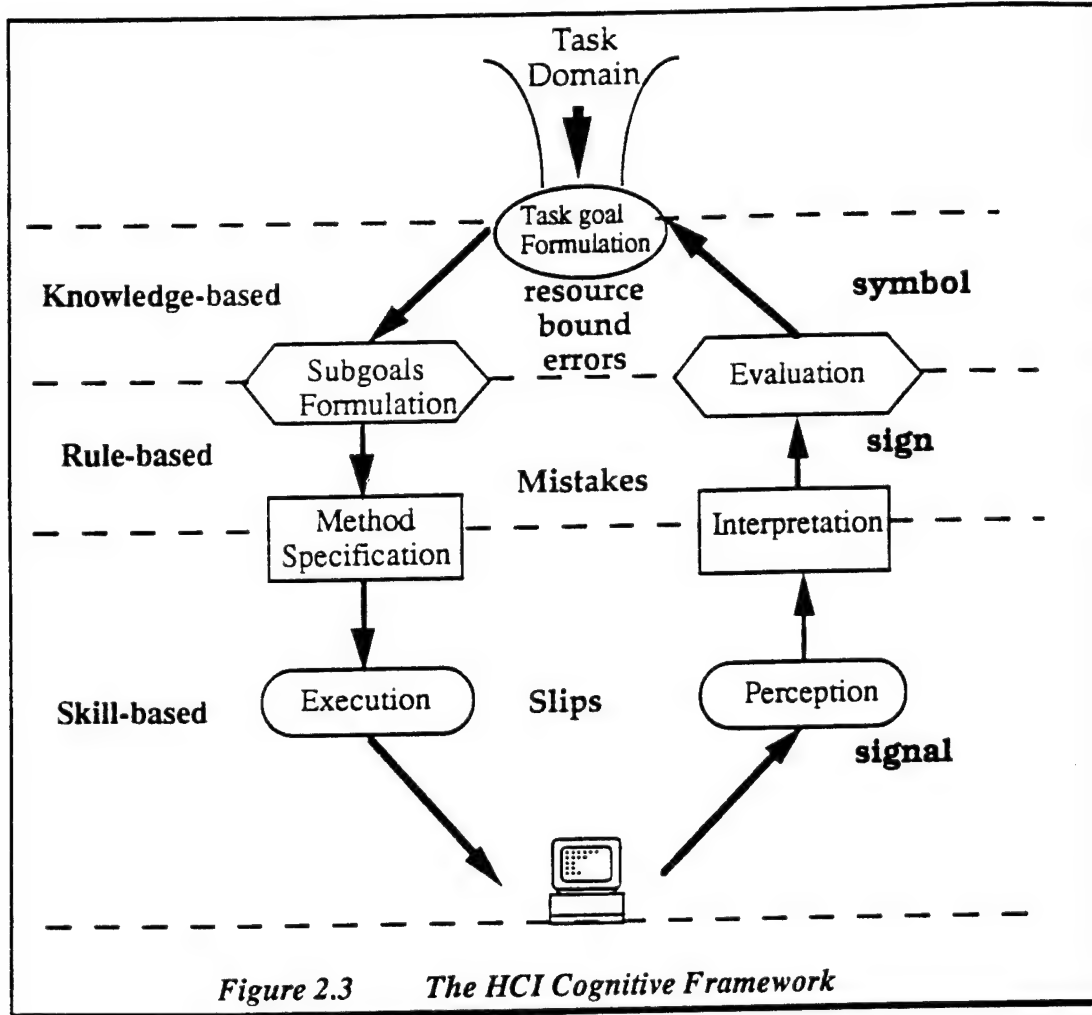
2.1.4   Types of Cognitive Control

The SRK framework gives us a guideline of the three different types of cognitive control that are possible, skill-, rule-, and knowledge-based, depending on the users' familiarity with the current task. In general, the user proceeds through the seven activity stages in the HCI framework in sequence. However, as stated by both Norman's Action Theory and Rasmussen's SRK framework, the higher level behavior may be skipped if the user is familiar with the task. That is, not all tasks will need knowledge-based level control as users might launch into rule-based or even skill-based behavior for well-practiced tasks. There exist short cuts to go from one stage directly to another non-adjacent stage. Also, stages may be repeated as necessary. For example, in using the

mouse to highlight a familiar menu item, the cycle of the skill manipulation of the mouse and the perception of the signal as the flashing cursor position is repeated until the cursor is moved to the desired menu location. In an example in which a user issues a familiar command to alter a system state, the knowledge-based cognitive control will be skipped and the user will launch into rule-based control to issue the entrenched command directly which translates into the skilled typing of the required keys. Thus, these three levels of cognitive control play important roles in different situations.

During knowledge-based performance in unfamiliar situations, the user employs his problem solving skills to map task goals into action subgoals that will bring the system closer to the desired state. A major task at this level of performance is to transfer those properties of the task to a symbolic representation understandable to the users and more amenable to problem solving in the system space. To arrive at the action subgoals, the user employs a strategy, e.g., analogy, mental models, generalization from other solution methods, data transformation, etc., to transform the symbolic representation of the task into subgoals with known execution procedures. The performance at this level depends on the semantic content of the task, the strategy, and data transformation used to achieve the task goal.

During rule-based performance, the user employs a known solution plan instead of problem-solving to arrive at the set of subgoals necessary to accomplish the task. For each subgoal, there is an associated method or procedure to execute it. The procedure could be long and complex for a familiar task. Rule-based performance depends on the syntactic form of the interface language and the ability to recognize the states and to recall stored rules.

During skill-based performance, the execution of the procedures is translated into well-drilled sensory-motor behavior like skilled typing and its tactile feedback, and mouse maneuvering and the cursor signal feedback from the display. This level of

performance depends on the physical form of the physical objects manipulated. The more fluid the users are at these physical tasks, the more the users are able to free resources to compose complex task sequences at the rule-based level. For example, if a user is not a skilled typist, instead of issuing one rule that says type the command '/wcs' (set column width in the Lotus spreadsheet program), he needs to formulate a series of rules to find the '/' key on the keyboard, then hit it, find the 'w' key, then hit it, and so on. The cognitive control will need to be even one-level higher at the knowledge-based level if the user is not familiar with the command to change the column width.

The Nature of Cognitive Control. Besides having different triggering conditions, the three levels of behavior have fundamental differences in the nature of the cognitive control. The cognitive control for rule-based and knowledge-based behavior is serial, slow, deliberate, resource-limited, and is accessible to consciousness. On the other hand, the cognitive control for skill-based behavior can be parallel, is rapid, effortless, not resource bound, and possibly not accessible to consciousness [Shiffrin & Dumais, 1981]. It is possible that through repeated practice cognitive control may shift from higher to lower levels.

The Transfer of Cognitive Control. The control of user activity shifts from knowledge-based to rule-based to skill-based as the familiarity with the task increases through practice. As the user increases his skill level, the size of the task fragments increases. During early skill acquisition, the activities are controlled by rules and feedback governing very elementary acts, e.g., pressing a single key, moving the mouse. As the skill develops, the feedback will be stimulus patterns and the rules will be procedures that execute a longer sequence of elementary acts. During fully-developed skill performance, the activities are controlled by automated routines without much need for conscious monitoring. Rasmussen [1986] stresses that it is not the behavior patterns of the higher levels that are becoming automated skills. Automated time-space skills get

developed and become more elaborate while they are controlled and supervised by the higher-level cognitive activities that eventually deteriorate. Users will eventually lose track of the basic causal or functional understanding when ski..s get fully developed. This has a sobering implication for interface design—bad interface can still produce skilled users if the users are pushed through enough rote practice.

All the above discussion about the three levels of behavior or cognitive control assume that the user gets the required feedback from the system appropriate to the current level of cognitive control. The following section develops the concept that the same feedback can be perceived as signals, signs, or symbols depending on the level of cognitive control.

2.1.5   Types of Stimuli Processed

The role of the information observed from the environment is different in the three levels of cognitive control. Users perceive **signals** during skill-based behavior, interpret **signs** during rule-based behavior, and evaluate **symbols** during knowledge-based behavior. The same event can be perceived as signal, sign, or symbol depending on the user's familiarity with and the role of the expected task feedback.

At the skill-based level, the control of the sensory motor system is continuous, synchronizing the physical activity such as navigating the menu using the mouse. During this level of control, the system feedback is perceived as time-space signals, continuous, and quantitative indicators of the system state. These signals have no meaning except as direct physical data. They are generally confirmatory in that they are expected. For example, during menu traversal using a mouse, the user perceives the feedback of flashing menu items from the screen to know that the mouse movement is indeed received by the system and the navigation is within the bounds of the menu. During skill-

based behavior, a change in the screen display or even the tactile feedback of keys pressed can be perceived as adequate feedback of successful execution.

At the rule-based level, the information perceived is defined as a sign. Signs are generally labeled by names that refer to system states or action subgoals, e.g., a particular display, a menu item, etc. Signs do not reflect the concepts or the functional properties of the environment; they only serve to activate well-learned predetermined procedures. Signs can only be used to select or modify the rules controlling the sequencing of skilled subroutines but they cannot be used for reasoning or to generate new rules. In our example of menu traversal, expert users look actively for the desired label of a menu item as a sign of an action that will accomplish a subgoal.

At the knowledge-based level, information must be evaluated as symbols for causal functional reasoning in order to predict or explain unfamiliar system feedback. Symbols refer to concepts relating to the functional properties of the task space that can be used for reasoning. Again using our menu traversal example, naive users will need to read the label of the menu item, compare it to his internal functional representation of the task and decide whether the menu item will accomplish the task goal or move the system state closer to the task goal.

2.1.6   Types of Errors

Many of the current HCI cognitive models, e.g., the GOMS family of models, do not model errors explicitly. However, human errors and their prevention, prediction and recovery play an important role in interface design. Analysis of human errors is important to validate HCI models. Success in the prediction and simulation of error-free performance is no proof of the validity of a model [Rasmussen, 1986]. Also, research has found that even experts commit many errors. In an expert text editing study, it is estimated that 35% of the expert's time and 80% of the variability in time can be

accounted for by errors [Landauer, 1987]. In another study, up to 21% of total tasks involve major errors [Roberts & Moran, 1982]. It is thus important to include error categories and their underlying cognitive process as part of our HCI framework.

There are also different types of possible errors associated with the different levels of cognitive control: **resource bound errors** are associated with knowledge-based behavior; **mistakes** are associated with rule-based behavior; and **slips** are associated with skill-based behavior.

The types of possible errors during knowledge-based behavior are caused by limited resources. The errors arise out of not knowing the methods to execute the tasks (limits of learning), applying the wrong analogy (limits of problem solving), etc. Also, errors can arise out of memory overload, keeping track of too much information in the working memory. In our HCI framework context, keeping track of multiple nested uncompleted subgoals contributes to memory load.

During rule-based behavior, errors are most frequently manifested as recall of incorrect rules or incorrect recall of rules. A high frequency rule may dominate a less used rule if the triggering conditions are similar. Another mistake is forgetting an isolated action in a long action sequence, e.g., forgetting to select a piece of text before issuing a formatting command.

During skill-based behavior, errors are termed "slips" as they are non-intentional and arise due to time-space coordination problems. For example, in menu traversal using a mouse to select an item, the user may select the wrong item because attention has already been switched to getting to the next task. Another example is the reversal of the order of two characters during touch typing.

Error Recovery

Just as higher level behavior serves as cognitive control for lower level behavior, higher level behavior also serves as cognitive control for the monitoring and recovery of lower level errors. A slip will need to be recovered by a rule-based or knowledge-based behavior. If there is a known rule to recover a slip, e.g., an UNDO command, that rule will be applied to reverse the undesired system state caused by the slip. If the slip brings about an unfamiliar system state, knowledge-based problem solving behavior must be initiated to recover the damage done. Similarly for mistakes, a remedy rule will be applied if one is known; if not, knowledge-based behavior will be needed. There are really no good remedies for resource bound errors when the user does not know the right procedures except to continue to explore for the right actions needed to accomplish the task. Working memory overloads are the worst kind of error as the only remedy is a complete redesign of the interface to avoid the excessive memory load.

2.1.7   Types of Memory Stores and Types of Memory Representations

Implicit to the above discussion are the types of memory resources needed and the types of memory representations for the different stages of activities. Skill-based behavior uses cognitive resources in the form of motor memory and visual memory. Rule-based behavior taps the long-term memory where the rules are stored. Short term memory may be needed to hold the expected feedback from the system for interpretation. Knowledge-based behavior requires both the long and short term memory. The user retrieves from the long term memory general knowledge needed to decompose the present problem. Short term memory is needed to plan and keep track of the action subgoals.

During skill-based behavior, the memory representation is a time-space motor program for skill-actions [Rasmussen 1976]. During rule-based behavior, the memory

representation is procedural knowledge. For knowledge-based behavior, the memory representation is declarative.

We have now described all the components in our HCI framework. In summary, to create our HCI framework, we have adapted and integrated Norman's Action Theory to describe the stages of activities and Rasmussen's SRK framework to describe the underlying cognitive processes when users interact with computer applications. We will next use this HCI framework to review the literature on HCI cognitive models.

2.2 Current HCI Cognitive Models and The HCI Framework

The HCI framework will help us structure the literature on HCI cognitive models. To keep the review within reasonable length, it will necessarily be brief. An outline of the models will be given but examples of the use of the models will be omitted. However, references to the original work are given where interested readers can find detailed descriptions and examples of use. After outlining the models, we will describe how the models fit into our framework.

Most current HCI models are rule-based and skill-based models. This can be expected as rule-based and skill-based behavior are predictable and more amenable to quantitative models. There is very little research on modeling the feedback loop of the framework, i.e., there is not much done to model users' perception of device information output. Lohse's [1991] cognitive model of perception begins to explore this area. There is also very little research on knowledge-based behavior—how users map task goals into action subgoals. Following is a review of the major HCI models in the literature. We will group the models under the headings of formal grammar family and the GOMS family of models. Under the formal grammar model and its variant, we will review Reisner's [1981] BNF model, Moran's Command Language Grammar (CLG) [Moran, 1981], and External Task to Internal Task mapping (ETIT) [Moran, 1983], and Payne

and Green's [1986] Task Action Grammar (TAG). Under the GOMS family, we will review the GOMS model and the keystroke level model (KLM) [Card et al., 1983], Keiras and Polson's [1986] Cognitive Complexity Theory (CCT), John's [1988] Critical Path Method (CPM) extension of the GOMS model, Lewis and Polson's [1990] CE+[2], and finally, Young and Simon's [1987] Planning Model.

2.2.1 Formal Grammar Family

The behavior of a human interacting with a computer can be viewed as a Language [Payne & Green, 1986]. Within this context, Formal Grammar is a tool that allows the structure of the interaction to be analyzed and manipulated. A Formal Grammar specifies the structures of a language by constructing a collection of rewrite rules that can be used systematically to generate each and every possible legal communication. When used in HCI by models like TAG, CLG, etc., these rewrite rules produce a series of action sequences made up of elementary tasks and cognitive units. An example of an elementary task is to type a letter and an elementary cognition unit is to retrieve a command.

Formal grammar models are competence models as they only model the legal task sequence that can be produced but not the actual performance in terms of time and error. The structure of the grammar does not allow easy prediction of actual performance in terms of the performance time, the underlying cognitive processes and limitations, and how users derive those rules. They are rule-based behavior models where each task goal is described by rules that specify the possible final action sequences. Ease of use of a

---

[2]Not an acronym. The name comes from the fact that CE+model is an integration of the CCT Theory and the EXPL model on learning from examples [Lewis et al. 1988].

device is measured by a complexity metric, normally a measure of the number of rules, the number of similar rules, and rules that can be transferred from other systems.

Reisner's BNF Grammar

Backus-Norm Form (BNF) notation has been used to formally describe programming languages using context-free phrase structure grammars. BNF allows the syntax of the language to be described precisely and the notation can be used to write compilers that are executable by programs. BNF can thus serve as a meta-language to describe different languages. Reisner [Reisner, 1981; Reisner, 1984] adopted this notation to describe the HCI 'language' where each task sequence can be viewed as a language statement.

Reisner describes the user's knowledge of the system in terms of the grammar of an 'interaction language'. The grammar describes all the rules which produce legal action sequences. According to Reisner, the resulting rules can then be used to predict performance time and errors according to the number of different terminal symbols, the lengths of the terminal strings for particular tasks, and the number of rules necessary to describe the structure of some set of terminal strings. However, such predictions are not very successful as analysts using this notation need to supply the time estimates for each terminal action themselves.

As cast in our HCI framework, the BNF grammar only models one stage of activity, the method specification stage. The cognitive control is rule-based and the grammar does not model the feedback loop nor the planning stage.

One criticism of the Reisner approach is that it does not model the consistency of the interaction language. For example, in the BNF notation, five rules with similar structures are as difficult to learn or perform as five rules that have totally different structures.

The Task Action Grammar (TAG)

Payne and Green [Payne & Green, 1986] chose to model HCI activities using a Task Action Grammar after observing that none of the existing models addresses the consistency issue of the interface. It is reasonable to assume that an interface that has a consistent structure or allows users to execute the tasks using objects and action sequences conceived in the task space will lead to better learning and performance. TAG is able to account for consistency of interface structure by using a two level grammar to account for rules that have similar syntax. When a family of tasks is executed by task sequence with the same structure, or with the same syntax using formal grammar terminology, only one rule is needed for the family of tasks.

TAG describes the task in terms of simple tasks and rule-schemata. Simple tasks are those that users can routinely perform that can be accomplished with no control structure, i.e., simple tasks are the subgoals in our framework. One example of such simple tasks is to move the cursor one character to the right. The rule schemata are production rules where the simple tasks are represented according to their features. This way, two tasks that have the same structure but only differ in the features can be represented by only one rule instead of two. For example, moving the cursor in any one of four directions has only one rule that has a direction feature that can take on one of four direction values. This is how TAG models consistency.

Like BNF, TAG does not attempt to model the cognitive process involved in HCI. It only describes what the user knows but not how the user uses that knowledge. How users combine several "simple tasks" to execute a task goal is not specified. There is no mechanism to chain simple tasks and there is no precise definition of what a simple task is. It does suggest that there is lesser learning and performance cost for novel simple tasks that share the same structure with known simple tasks. In the authors' words, TAG models family resemblance and semantic consistency.

TAG covers almost the same aspects of our HCI framework as BNF except that TAG models the user's knowledge of the structure of the interface language. It again uses rule-based cognitive control for specifying action sequences but adds the knowledge-based control of the understanding of the task structures. Users can apply generalization to infer methods to execute novel simple tasks that share the same structures with known methods for executing other simple tasks. Errors are modeled only in terms of applying the wrong features due to inconsistency in language structure and semantics. For example, in a system where the command to delete a word is DELETE WORD, users would guess the corresponding command to delete a line as DELETE LINE, not REMOVE LINE.

Both TAG and BNF are rule-based models as they map external tasks directly to internal operations. There is no role for subgoals in these two models. As cast in our HCI framework, both TAG and BNF model only one stage of activity, the method specification stage. The cognitive control is rule-based and the models do not model the feedback loop nor the planning stage.

Command Language Grammar (CLG)

Command Language Grammar (CLG) [Moran, 1981], takes an approach that uses several levels of mappings with the potential to model the different levels of cognitive control. Instead of one direct mapping, CLG offers a hierarchy of mappings. The levels of description are designed to correspond to the levels of representations that users presumedly have. The grammar consists of three components and each component has two levels: conceptual component (task level, semantic level), communication component (syntactic level, interaction level), physical component (spatial layout level, device level). The last two levels, the spatial layout level and the device level, are not described in the model explicitly.

Grammar rules like those in BNF are used to describe knowledge in each mapping level. The task level describes the task domain addressed by the system. The semantic level describes the concepts represented by the system. The syntax level describes the elements of the interaction language in terms of commands, arguments, context and state variables. The interaction level describes the actual interaction between user and system: the keystrokes, mouse movement and system feedback.

The different levels of description in CLG correspond to the different stages of activities in our HCI framework. The task level can be equated to the task goal formation stage; the semantic level is the action subgoal formation stage; the syntax level is the method specification stage; and the interaction level is the execution stage. Like our HCI framework, CLG states that users need not have knowledge at all levels in order to use the system. Some lower level knowledge will be procedural while other knowledge can be declarative. Users can operate with just the interaction level knowledge but will not have the necessary system concepts for error recovery. On the other hand, users may know the task goals but not know the method nor the commands to reach it.

CLG is again a competence model and not a performance model. No time parameters are provided for the interaction level elements. The model does not address users' cognitive limitations and does not model errors or the processing of feedback. The CLG approach seems potentially powerful as it covers many aspects of the HCI framework but it is too complicated to put into practical use. Another drawback of the model is that it does not show how to map from one level of description to the other.

External Task to Internal Task (ETIT) Mapping

Moran [Moran, 1983] developed External Task to Internal Task (ETIT) Mapping to address some drawbacks of CLG. ETIT is simple to use and it provides a mapping between the two levels of description. ETIT is not strictly a cognitive model because

users' cognition is not modeled but implied by the complexity of mapping from the external task space to the internal system space. ETIT is a form of task analysis that models the mapping between the users' representation of the task and the system's representation of the task. ETIT compares representations of these two spaces. Each external task is represented by an action and an entity as defined in the task domain. The internal task is also represented by an action and an entity but defined in the system domain. The number of rules needed to map the external task space to the internal task space reflects the complexity of the system. Moran also suggests that transfer of knowledge from one system to the other can be measured by the common rules between the two systems.

When cast against our HCI framework, ETIT is very much like Reisner's BNF grammar as it only models the method specification stage. The cognitive control is rule-based. Although a promising idea due to its ease of use, ETIT has not been used in practice or further developed.

We have now reviewed the major HCI cognitive models based on formal grammar specification or its variants. These models generally do not model the user's cognitive processes, cognitive limitations and erroneous behavior. They are all rule-based cognitive control as a result of the rule-rewrite structure of formal grammars. We will now turn our attention to another family of models, the GOMS models, which enjoys better success and acceptance than the grammar model due to the ability to predict performance behavior and time.

2.2.2 The GOMS Family of Models

The GOMS (Goals, Operations, Methods, Selection rules) family of performance models is the most well known and discussed in the literature. Many extensions and examinations of the GOMS model have surfaced in the literature (see Olson and Olson

[1990] for a review). Its success has spurred the growth of the cognitive engineering paradigm in HCI research. The GOMS model, and others that follow this approach, predicts users' performance by modeling the users' knowledge of the operations to complete a task. Using the users' knowledge structure and a set of performance parameters provided by an underlying cognitive architecture , GOMS models make reasonably accurate performance prediction. The underlying cognitive architecture is called the Model Human Processor (MHP). The most widely used of the GOMS models is the Keystroke Level Model (KLM) where the elementary operations are those that have the same time magnitude as that of pressing a keystroke.

GOMS was intended only to model the flawless performance of expert users. The model's prediction is reasonably accurate when restricted to expert users but because it avoids modeling errors and non-expert users, it has been much criticized. Although it has its share of limitations, it has nevertheless served as the benchmark against which other models are compared. Early applications of GOMS analyses have been restricted to limited domains, e.g., text editing and system commands, but have now been proven useful in other domains like spreadsheet [Olson & Nilsen, 1988] and telephone operator workstation software [John, 1990].

We review the GOMS model, its companion Model Human Processor (MHP), as well as related models including the Keystroke-level Model (KLM), the Cognitive Complexity Theory (CCT), the Critical Path Method (CPM) extension to GOMS, Polson and Lewis's CE+, and the Planning Model. Again, the review will be necessarily brief but references to the original articles are given. Furthermore, readers can refer to Olson and Olson [1990] for a comprehensive review of the GOMS family of models and their extensions.

The GOMS Model

The GOMS model describes the cognitive control in terms of the knowledge users must have to interact with the system. The knowledge is described in four categories: Goals, Operations, Methods, and Selection rules. The initial letters for these four categories of knowledge make up the acronym GOMS. The Goals (or Subgoals) are the system states or task objectives the user wants to reach. The Operations are the cognitive and physical actions when using the system. The Methods specify combination of actions that form a procedure to achieve the goals or subgoals. The Selection Rules specify which methods to choose to achieve a goal given a certain task condition. The goal structures, the method choices and the selection rules together describe the cognitive control for the operations.

The GOMS model is actually a family of models where each deals with a different grain size of analysis. The grain of analysis depends on the grain size of the operations described. At the one extreme, the operations are unit-tasks which accomplish subgoals. This level of analysis is useful for initial system design where the available elementary operations are not known. However, the time estimate for the unit-task level operations will necessarily be coarse thus diminishing the predictive power. The unit-task GOMS models seldom are used in practice.

At the other end of the spectrum, the operators are defined in terms of elementary operations where time estimates are available from the MHP model. The level of analysis is called the keystroke-level model (KLM) as the time estimates for the operations are the same magnitude as the time to press a keystroke. The KLM proves to be the most widely-used GOMS type model as it can predict performance time to a fair degree of accuracy for expert users. We will describe the keystroke-level model in the next section.

Although the GOMS models seem to provide the mechanism of cognitive control to cover all three levels of behavior in our HCI framework, in practice so far it is basically a rule-based model. The potential for using the goal structure to describe knowledge-based behavior has not been explored in detail. Card et al. [1983] specifically state that goals do not contribute to the time calculations of the various models since goal manipulation should not take more than .5 seconds (page 182). The subgoals are used only to break the task into subtasks called 'unit-tasks' where known methods of operations are available to the users. In the GOMS world, Goals and Operators are not really different sorts of things. They both refer to actions taken by the users, of greater or lesser scope. Operators are simply Goals which are treated as elementary at a particular grain of analysis [Simon & Young, 1988]. In GOMS analyses, all goals or subgoals are fully decomposed into a set of elementary operations for execution. Thus, the GOMS model is not much different from the formal grammar model previously described. What makes the GOMS models different from the grammar models is the parameterization of the elementary operations that gives the GOMS models ability to estimate performance time. Also, the selection rules to choose among multiple methods that can be applied to the subtask are the cognitive control in the GOMS models not present in formal grammar models. Otherwise, like formal grammar models, the specific GOMS models explored to date do not address the details of knowledge-based behavior in unfamiliar situations, do not address how users handle errors and the feedback other than signals from the system.

The Model Human Processor (MHP)

The Model Human Processor (MHP) [Card et al., 1983] is not a GOMS model but it defines the cognitive architecture of the human information processing system that is common to all GOMS models. This cognitive architecture is defined by a set of operating principles for the cognitive system constituting the cognitive, perceptual, and

motor processor and their associated stores. The operating principles are a succinct summary of the human performance literature, e.g., Fitts' law, problem space principle, etc. They describe the possible behaviors under different conditions. However, these principles are not referenced in the current GOMS analyses. What is used in the GOMS analyses is the set of parameters for the elementary operations which Card et al. [1983] abstracted from the literature. Examples of these elementary operations are: the time to retrieve something from short-term memory, the time to press a key on the keyboard, the time to get a piece of information from the screen, etc.

The MHP serves as a foundation for the rest of the GOMS family of cognitive models by providing a common cognitive architecture and performance time parameters. The GOMS family of models simply provides a knowledge control structure that defines how the users will interact with the device with operators of different grain sizes. If the operators are in the grain size of the elementary cognitive, motor, and perceptual operations, we obtain the keystroke level model. The KLM can estimate performance time with reasonable accuracy by summing up the time estimates of these elementary operations provided by the parameters of the MHP. Card et al. [1983] called the parameterization process "simplifying theory into practical engineering models."

The parameterization in GOMS analyses fits best when the users are experts engaging in routine and flawless behavior. The MHP does have a small bandwidth for each parameter to vary in terms of the performance for slowman, middleman, and fastman. This bandwidth is too small to address non-skilled, non-routine or erroneous behavior.

In terms of our framework, the MHP parameters occupy the space of skill-based behavior. They basically summarize the time needed to perform activities at the skill-based level of behavior and the time needed to perceive signals from the display. These parameters will provide the time estimates needed for the rule-based GOMS family of

Models. The operating principles in the MHP has the potential to address skill-based behavior but they are not used in current GOMS analyses.

The Keystroke Level Model (KLM)

The KLM is the best known in the GOMS family of models. It predicts performance time of expert users executing familiar tasks that have been developed into routine cognitive skills. In the Original conception, experts would need little thinking time and their action times would be less variable than those of novices. In KLM, the description of the user's task contains only the sequence of effective elementary operations described in MHP without parameters reflecting the control structure like goals and methods. In this original formulation, there is only one mental operator (estimated to take 1.35 seconds) to account for all cognitive operations like memory retrieval, mental preparation, etc. The model has a set of rules for when to apply a mental operator to the keystrokes, typically in front of each cognitive unit approximated by a command string.

The KLM as cast in our HCI framework will occupy both the rule-based and skill based behavior spectrum. The rule-based behavior is the retrieval of unit tasks and their associated operations. The skill-based behavior is translated into performance time for skilled operations by the MPH model level parameters.

Currently, the KLM achieves its degree of accuracy for predicting performance time by having some highly restrictive assumptions. First, the model only applies to expert users. Second, the task acquisition time and evaluation time of unit tasks are not modeled. In the GOMS models, tasks are broken down into more manageable subtasks called unit tasks. The time to acquire and evaluate the completion of such unit tasks are not modeled in the current models. Furthermore, there are two assumptions of the unit task performance time as stated by the authors: (1) the execution of the unit-task is

assumed to be the same no matter how the unit-task is acquired: and (2) acquisition time and execution time are assumed to be independent [pg 261 Card et al., 1983]. These two assumptions are only reasonable when users have no problem decomposing the tasks into unit tasks like those expert users modeled in the GOMS models. For non-expert users or complex tasks, the execution is dependent on how easily the tasks can be decomposed into unit tasks. Finally, the KLM model also assumes that all elementary operations must be serially executed.

Such restrictive assumptions brought many criticisms to the KLM model. However, the KLM proves promising as it allows prediction of performance without resorting to empirical evaluation of finished products. It spearheaded the cognitive engineering approach in HCI research and there is much further research that builds on the KLM model. These later models extend the original KLM by examining the KLM assumptions. The following two models extend KLM by attempting to address some of these restrictive assumptions by explicitly modeling novice learning, parallel operations, and memory load.

Cognitive Complexity Theory (CCT)

In the Cognitive Complexity Theory [Kieras & Polson, 1985; Polson, 1987], the knowledge represented in the GOMS model is formalized as a production system. There is also a representation of the device in the form of a Generalized Transition Network (GTN). The device knowledge component has been dropped in subsequent use of CCT. By coding the knowledge in GOMS as a production system, CCT allows quantitative prediction of training time and transfer of training by counting the new production rules a user needs to acquire. Kieras and Polson estimate that it takes 30 seconds to learn a new production rule. The CCT also models explicitly the working memory for holding active goals or subgoals. The depth of the goal stack reached during task execution is taken as

an indicator of the task difficulty. The use of production systems to model memory load is promising; Lerch [1988] and Smelcer [1989] were able to use it to model the user's working memory load and errors in tasks involving keying financial formulas and database query, respectively.

The main criticism of the CCT approach is that it derives its predictions solely from the amount of knowledge but not the type or content of knowledge in the model [Knowles, 1987]. The model thus will not be able to model the user's ability to generalize a new command that shares a common structure with some learned command; this new command still corresponds to a new production rule. In TAG parlance, the model cannot account for the interface consistency issue. Also, the model does not address how different types of subgoals , e.g., a subgoal that arise naturally due to task decomposition versus one that arise as an artifact of having to use the interface, will contribute differently to the memory load. Like GOMS, the subgoals in CCT are still not a cognitive control structure but just a notation as a placeholder for the elementary operations. Also, the feedback portion of the HCI framework is not modeled.

CCT as seen in our HCI framework is very much like the GOMS models; it is a derivative. CCT adds the cognitive control of a working memory and models knowledge-based behavior involved in learning new methods. What it does not address are problem solving behaviors for exploring new commands and how subgoals are formed.

Critical Path Method Extension to GOMS

John [1988] extended the KLM assumption that the elementary cognitive, motor, and perceptual operations as serial. The assumption of serial operations is a reasonable assumption in KLM to simplify the process of estimating the total time by adding all the times of the operations in a unit-task. However, John argues that the prediction of the

time of highly skilled performers can be improved if this assumption is relaxed. In the task of expert transcription typing investigated by John, the expert can be expected to look ahead to the information for the next piece of task while still typing on the current task. John uses Critical Path Method (CPM), a familiar tool in engineering operations research, to model the possible parallel or cascading dependencies among the mental, motor, and perceptual operations.

In our framework, CPM takes the same position as the KLM except that the prediction of skill-based performance time is no longer calculated by adding up the KLM parameters. It is now necessary to use critical path analysis to model the underlying elementary processes to see where elementary operations overlap.

Polson and Lewis's CE+

The CE+ model [Polson & Lewis, 1990] attempts to model exploratory learning in HCI by integrating theories from (1) the GOMS model and CCT on the representation of procedural knowledge as productions, (2) the EXPL model on learning from examples [Lewis, 1988], and (3) research on problem-solving processes. This is the first model that we have reviewed so far that attempts to address knowledge-based behavior in unfamiliar task environments and to incorporate the feedback from the system in the model.

The EXPL model was developed to account for the role of feedback in learning procedures. The model assumes that the user engages in a causal analysis of a sequence of user actions and system response to determine whether the response could be attributed to the actions. The causal reasoning is accomplished by a set of simple heuristics; for example, the identity heuristic places a causal link between action and response if both share the same verbal element.

In the problem-solving component of CE+, a "label following" heuristic is used to predict naive users' exploratory behavior. This heuristic states that when faced with a

choice among alternative actions, users will choose the action whose description overlaps most with its goal, provided the action has not been tried unsuccessfully before. Once the actions have proven to be successful for the goal, they will be compiled into a production rule in the CCT fashion.

The CE+ model is in its developmental state but it is promising as it focuses on the knowledge-based behavior not addressed by other HCI models. The current CE+ model produces descriptive guidelines for design for ease of learning but not quantitative predictions of learning time. The immediate compilation of successful exploration into production rules seems to suggest there is no middle ground between problem solving behavior and skill-performance. The integration among the various components of the models also needs further refinement.

The Planning Model

The last model we will review is a model that integrates the planning and skilled procedure execution in HCI. Young and Simon [1987, 1988] propose that planning processes in HCI are different from those modeled in the artificial intelligence literature where a complete plan must be generated after searching the problem space. They suggest that during planning in HCI, the activity of planning is interleaved with the execution of the plans (which may be partial); and that simple, partial plans are more appropriate than complex, detailed ones. They claim that such an approach to planning can model behavior that spans our HCI framework spectrum, from problem solving behavior to the smooth execution of routine methods.

Their model is similar to the GOMS model in that it is hierarchically described by goals and operators. How the model differs from GOMS is that the decomposition of the hierarchical goals needs not be complete. Those goals with unknown methods of execution will not be expanded into operations. When users encounter such goals, they

will engage in a problem solving behavior like the users modeled in CE+. For fully decomposed task goals, the Planning model acts like a GOMS model.

The planning model again attempts to step beyond the bound of a rule-based GOMS type model by considering knowledge-based behavior as described in our framework. This model, like CE+, is descriptive for its present incarnation and leaves the underlying cognitive process of the planning mechanism unspecified. Unlike CE+, it does not suggest how search in the problem space can be carried out to find the operators for unfamiliar tasks.

We have now reviewed the major HCI cognitive models. Almost all models reviewed above address rule-based and skill-based behavior with the exception that the last two models venture into investigating knowledge-based behavior. The next section describes how the human behavior literature in general fits into our HCI framework to reveal the gap in the HCI cognitive modeling literature.

2.3   The HCI Framework and the Human Behavior Literature

Our HCI framework reveals that the current HCI cognitive models only tap a small portion of the available human behavior theories. These theories come from the literature in many related fields, for example, psychology, human factors, artificial intelligence, linguistics, etc. Figure 2.4 shows some theories from the various fields that HCI research can draw on as seen in our HCI framework. The list of theories is not exhaustive and we will not review those theories here. The list is just an indication of the vast human behavior literature that HCI research can draw on. It can help both interface designers and HCI researchers to frame their problem space and draw on theories from related fields for those areas that current HCI models do not address.

Figure 2.4 The Human Behavior Literature and the HCI Framework
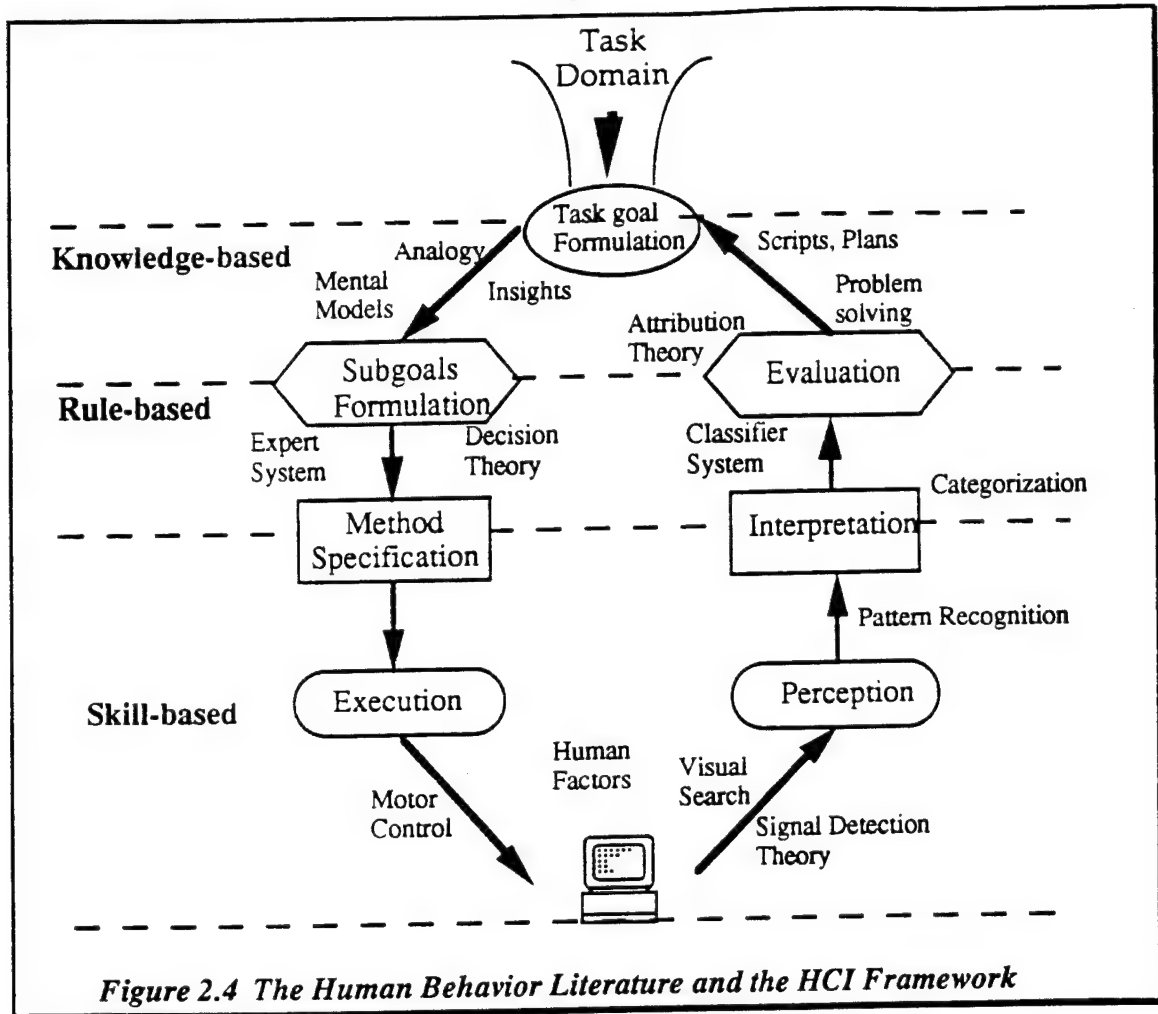
In the Knowledge-based domain, we can draw from research findings in problem solving and AI. Problem solving literature [Newell & Simon, 1972] provides us with theories on how people use mental models, analogies, insights, etc., to arrive at novel solutions in the HCI domain. Theories of Scripts and Plans in the AI literature [Schank & Abelson, 1977] inform us how a user can break a long task sequence into more manageable subtasks. Attribution theory [Kelley, 1973] can inform us how users can use the feedback for causal reasoning to see whether a subgoal has been accomplished by a certain method in a novel situation. Tversky and Kahneman's [1974] work on decision theory can tell us how users might use different heuristics to tackle an unfamiliar task.

In the rule-based behavior domain, the AI literature provides us with expert systems [Shapiro, 1987] and classifier systems [Holland, 1986] that can shed light on how different methods are chosen for certain tasks. The pattern recognition literature [Sekular & Blake, 1985] can inform us how signals are processed into signs that can trigger a rule-based response. If HCI activities can be seen as communication between the device and the user through an action language, then the linguistic literature, in particular psycholinguistic literature [Slobin, 1979], can provide research findings in sentence production and comprehension to help us assess the difficulty of forming the action sequence specified by the method.

In the skill-based behavior domain, the human factors literature has the most to offer. Motor control literature [Nilsen, 1991; Schmidt, 1982] will inform us how users are affected by the physical design of devices. Signal detection theory [Swets, 1964] can tell us how users may perceive useful feedback from background noise. The literature on visual search [Cornsweet, 1970; Lohse, 1991] can also inform designers how to create an interface that facilitates quick identification of relevant information.

The above discussion cannot do justice to the wealth of knowledge available that is relevant to HCI issues. However, given the HCI framework we have developed, we can place any literature on human behavior in our framework and see how it can be used to describe the relevant stages or levels of HCI activities. Given this wealth of human behavior literature, the HCI cognitive models reviewed seem to cover only a portion of the literature and only some stages of the HCI activity cycle.

2.4   The Gap in the HCI Modeling Literature

The HCI cognitive models reviewed above, the formal grammar family and the GOMS family of models, cover primarily rule-based and skill-based behavior in our framework. There is certainly a need to explore more of the knowledge-based behavior.

Also, the feedback loop portion of the framework has not been well addressed by the HCI models. This is not a criticism of the HCI research but an indication that there is much room for the maturing HCI research to grow. In fact, the success of the GOMS family of models, especially the success of the keystroke-level model indicates that the HCI cognitive modeling approach is viable and that we can build on it. This is exactly what the last two models reviewed, CE+ and the Planning Model, try to do.

We will do the same in this research by proposing a theory that builds on the GOMS model by extending GOMS to include knowledge-based behavior. It is important that HCI models can address non-ideal expert behavior as we have argued that (1) even experts make mistakes; and (2) as computer applications get more varied and complicated, the true dedicated single-application expert user will be hard to come by. By investigating knowledge-based behavior, we will be able to show (1) how novices learn a new piece of software; (2) how users execute a complicated sequence of actions that are not amenable to rule-based behavior; and (3) which stages of the activities take the longest time. We proposed to do all these by investigating the roles of subgoal formulation as a cognitive control structure in our HCI framework.

2.5   The Roles of Subgoals in the HCI Framework

We will investigate the role of subgoals in our HCI framework as a cognitive control structure for knowledge-based behavior. This section starts by explaining why the subgoal structure has not been a major focus in the GOMS family of models to date. We then expound the possible roles of subgoals in our framework. Lastly, we propose a theory stating that a bad interface will force users to create extraneous subgoals that are difficult to acquire, maintain, and verify. We will show that it is these extra subgoals and not the number of keystrokes that account for the large differences in performance times and errors.

2.5.1 The Role of Subgoals in the GOMS Model

The role of subgoals in the GOMS model is de-emphasized as GOMS only models expert users performing routine tasks. In the review of the GOMS model, we pointed out that goals (and subgoals) do not contribute to the time calculations as they are used for notational purpose only since all goals and subgoals are fully expanded into their elementary operations. Time estimation is the sum of all the fully decomposed elementary operations time, and complex goals manipulation time is ignored. Also, the tasks modeled have a simple action sequence for each task goal and do not require problem solving or keeping track of complex subgoals sequence. That is, the need for using subgoals to break down complex operation sequences in these tasks is minimal.

In CCT, subgoals have a slightly more significant role as uncompleted subgoals are stacked in the working memory. The depth of the working memory is used as a gauge for the difficulty of a task. The extension is promising as Smelcer [1989] and Lerch [1988] were able to use this concept in modeling users' errors. However, the role of subgoals is still minimal in CCT as it is not used for modeling knowledge-based behavior. The aspect of learning that is modeled in CCT is a rule-based model, the time to learn is predicted by counting the number of new rules that must be learned. Subgoal structures are not used in modeling novice learning in CCT.

The unit task is a concept in GOMS, similar to the concept of subgoals in our HCI framework, which the authors define as a subtask that can be performed by the user within his performance limitations. The unit task is used as a control structure such that users have a method to execute each unit task formed. Our subgoal concept expands on this discussion by expanding the role of the unit task to include other possible roles of subgoals in knowledge-based behavior for cognitive control, e.g., the role as checkpoints for problem solving, the role as a bridge between the task space and the system space, etc.

2.5.2   The Roles of Subgoals in the HCI Framework

The primary role of subgoals in human behavior is cognitive control. It allows the use of limited cognitive resources to tackle difficult tasks. Human performance is bound by the limited working memory capacity and the limited capacity to attend to one conscious activity at one time. As such, human behavior is goal oriented and can be characterized in terms of goals and plans [Black, Kay, & Soloway, 1987]. The plans are the subgoals or procedures that will accomplish the task goals. Subgoals serve as check points against which progress towards the task goal can be measured. There should be pauses at the boundaries of these subgoals while the users evaluate what they have done and determine what the next relevant subgoal is. In the current GOMS models, these plans and subgoals are fully decomposed into elementary operations without much cognitive cost. We will discuss the role of subgoals in situations where there is substantial cognitive cost in acquiring and verifying subgoals especially when there is a wide gap between the task space and the system space.

To Segment a Complicated Action Sequence. In our HCI framework, the subgoals come between the task goal specification stage and the method specification stage. When the method is fully known and simple enough to be fully decomposed without overloading working memory, there is no need for the subgoals to act as cognitive control. When the method is complicated and long, the subgoals will be used to break the long action sequence into manageable subtasks where the user knows a method for each of these subtasks and they can be performed without exceeding working memory load. These subtasks are equivalent to the unit task in GOMS and the simple task in TAG where there are no cognitive controls within the subtasks to accomplish the subgoals. Subgoals allow the execution of a complicated task in the correct sequence by forming subgoals in a corresponding sequence.

To allow for non-optimal problem solving behavior. Subgoals provide the cognitive control for problem solving behavior when users do not know the exact method for execution. Here the subgoals are planned on-line rather than retrieved like those in the GOMS model. Subgoals provide a means to segment the task goal into many intermediate states where the user may know the execution method or can explore the operations to bring them to the desired intermediate states. This allows users to have partial plans instead of complete plans to achieve a task goal. Partial planning allows trial and error exploratory behavior when using an unfamiliar interface. The subgoals act as checkpoints to evaluate the expected outcome of the subgoal that will bring them closer to the task goal. The subgoals also act as checkpoints for error recovery and noting which methods are unsuccessful for a particular subgoal [Card et al. 1983]. If a certain method tried does not achieve the desired outcome for a subgoal, that method will not be used again for that subgoal and an additional subgoal needs to be initiated to reverse the outcome of the unsuccessful method.

This subgoal role of facilitating problem solving or exploratory behavior has an important implication for interface design. To allow for easy problem solving, the interface must provide clues or means to quickly constrain system search space so that the user can get on the right track. For example, menu systems that provide labels that match parts of the users' description of the task goal will facilitate the "label following" heuristics [Polson & Lewis, 1990]. An interface should also provide obvious feedback such that users know which state they are in and whether a subgoal has been successfully accomplished. Also, the interface should provide many "UNDO functions" so that users can reverse the effects of unsuccessful trial easily.

To bridge the task space and the system space. This last role of subgoals that we will discuss has the most impact on interface design. In a badly designed interface, the subgoals are used by users, experts or otherwise, to bridge the gap between the task goals

and the allowable system actions; i.e., to bridge the gap between the task space and the system space. This is akin to the Semantic Gulf in Norman's Action Theory [Norman, 1986]. There are two situations in which users need to create extra subgoals that are not a direct requirement of the task but emerge as a result of having to use the interface. First, when the task space and the system space operate on different object structures, there will be a need to create extra subgoals to bridge the differences. For example, if the task space operates on two dimensional objects but the system space only operates on one dimensional objects, there will be a need for using extra subgoals in the system space to operate on each dimension in the task space in turn. Second, when the system space does not allow users to carry out actions in an order as conceived in the task space, extra subgoals will be needed to buffer the out of order operations. For example, if users always think of creating an object first before defining its dimension in the task space but the system space requires the reverse operations, users will need an extra subgoal to temporarily stack the subgoal of creating an object first. The third use of subgoals already discussed is to bridge the system space and the task space by segmenting a long task sequence. This is not necessary a design fault but just a task space artifact that a task goal requires many steps to complete. The interface can be designed in cases such that each subgoal state is easily accomplished and verified, or to provide higher level operators that can accomplish a few subgoals at once.

The use of extra subgoals to bridge the task space and system space has not been discussed much in the literature. It touches on the issue of interface consistency, but not just within-interface consistency such as that modeled by TAG, but consistency between how the user thinks of the task and how the system allows the user to think of the task. The extra mapping required is the bottleneck in the HCI stages of activities especially when the extra subgoals are difficult to acquire and to verify. It is the number of

extraneous subgoals, not the number of keystrokes, that makes an interface difficult to use. This is the essence of our theory that will be verified by empirical studies.

The use of subgoals as a cognitive control structure to extend the GOMS model to include knowledge-based activities will address some of the criticism of the GOMS model. It will also build on the success of the rule-based GOMS model by extending the model to address non-optimal knowledge-based behavior.

2.5.3 The Use of Subgoals to Address Some Unresolved HCI Issues

We can use subgoals as a cognitive control structure to address some shortcomings of the current GOMS models. Some of these criticisms were brought up by Olson and Olson [1990] in their review of the GOMS models and some issues were brought up by other researchers and even Card et al. [1983], the authors of GOMS.

The planning, decision, and perceptual aspects of behavior were not modeled in the original GOMS model [Olson & Olson, 1990]. The subgoal structure will help to model these aspects by investigating the ease with which the interface allows the user to formulate the subgoals and to evaluate the feedback to check the accomplishment of the subgoals.

GOMS models only expert optimal behavior [Carroll & Campbell, 1986]; GOMS does not model erroneous behavior [Card.Moran, & Newell, 1980]. The addition of subgoals as control structure specifically addresses these vague issues by including knowledge-based behavior in the analysis. Issues like exploratory learning for novice and non-optimal expert behavior due to bad interface design can be addressed when the subgoal structure is considered. Subgoal structure explains why errors occur and provides a means to recover from errors. Errors can be explained by excessive memory load, bad feedback, and an interface that does not allow one to narrow the search path to

find the solution quickly, i.e., the operations or methods that bring one closer to task goal are not obvious.

It has been claimed that working memory load could be the major downfall of an interface [Waern, 1989]. GOMS does not explicitly model memory load [Card et al., 1983]. It is also not clear what contributes to mental workload [Olson & Olson, 1990]. In our formulation of the role of subgoals in the HCI framework, working memory load grows with the extra subgoals necessary to overcome unnatural mapping between the task space and the system space. The memory overload occurs when many uncompleted subgoals must be kept in the working memory.

KLM assumes that acquisition time and execution time of unit tasks are independent. Card et al. [1983] state that reducing execution time by making the command language more efficient does not affect acquisition time. However, our subgoal formulation argues that if an interface allows formation of a subgoal structure that reflects the task structure, the subgoal (unit task in the GOMS context) acquisition time will be greatly reduced.

Green [1988] points out that current GOMS cannot explain why a consistent interface will translate to better performance. Green's TAG only models internal task consistency. With the notion of subgoals, we can model both within interface consistency and external-to-internal task mapping consistency. The latter is even more important as it models how easily the user can translate his intention into allowable system actions. The consistency of the subgoal structure to the task structure is a measure of the external-to-internal task mapping consistency, the semantic gulf of the interface in the Action Theory. Norman explains what semantic gulf is in his Action Theory and our formulation of the subgoal structure explains the cognitive mechanism that caused this semantic gulf.

GOMS does not explain why small differences in interfaces often cause large differences in usability even if the number of key strokes remains about the same [Carroll & Campbell, 1986]. The current keystroke level model seriously underestimates the actual time [Allen & Scerbo, 1983]. The GOMS model focuses on quantitative tradeoffs and thus misses out on the many discontinuities in usability effects. Our theory explains these by stating that although the number of keystrokes may remain about the same, some interfaces will force users to use extra subgoals to overcome the bad interface. The acquisition and verification of these extra subgoals answer the question on which activities take the longest time. It is the time for planning, acquiring and verifying these extra and unnatural subgoals that overwhelms the keystroke time. We will illustrate this with two empirical studies.

2.6   Proposed Theory and Empirical Validation

This research has done the following in arriving at the theory of the role of subgoals in mapping external task space to internal system space. We have developed an HCI framework that imposes a cognitive architecture on the stages of HCI activities. The framework is used to survey the HCI literature and we find that there is a need to investigate knowledge-based behavior. We propose to investigate the subgoal structure as a cognitive control mechanism that can explain many knowledge-based behaviors like learning, erroneous behavior, etc. The rest of this research will investigate the role of the subgoals as a bridge between the external task space and the internal task space which has not been investigated in the literature.

2.6.1   The Subgoals as Bridge between the Task Space and System Space

We have explained in the previous section how subgoals can function as a bridge between the task goal and the eventual action sequence needed to achieve this task goal.

We will summarize here. (1) When the action sequence required to accomplish the task goal is complex and long, subgoals serve as an intermediate check points to segment the long sequence so that each segment has a corresponding method and can be performed within working memory restrictions. (2) When the task space and system space operate on different objects, extraneous subgoals will be needed to map from one space to the other. (3) When the system space does not allow the user to execute the action sequence conceived in the task space, extra subgoals will be needed to stack and monitor the out of order operations.

Our theory then claims that it is the number of these extra subgoals created by the need to bridge the system space and the task space that can account for qualitative differences between two interfaces. Extra subgoals will explain why two interfaces that produce nearly equal numbers of keystrokes can produce very different performance times and error rates. This discontinuity in usability effects is not modeled by the current KLM. The difference in the extra subgoals needed to overcome a bad interface can also account for the cause of the memory load. Finally, the acquisition and verification of these extraneous and unnatural subgoals account for the largest portion of the performance times. If the acquisition and verification are not accounted for in a KLM, the time prediction will vastly underestimate the actual performance time. On the other hand, if the acquisition and verification of subgoals are to be ignored like in the current KLM, the interface should be designed to reduce unnatural mapping, thus reducing the extra subgoals, eliminating the long pauses for their acquisition and verification. Also, with an interface that feels 'natural' to users where the objects manipulated and action sequence are those conceived in the task space, the need for planning (acquiring unnatural subgoals) and verification will be greatly reduced. The next section outlines two empirical studies that illustrate the above theory.

2.6.2 The Empirical Studies to Validate the Roles of Subgoals

The two empirical studies will show that the bottle neck of task execution occurs when the interface does not allow direct mapping of the task goal into corresponding subgoals that reflect how the user thinks of the task in terms of the objects to operate on or in terms of the order of operations. These unnatural mappings cause formulation of extra subgoals that results in long performance time and errors. The following paragraphs describe the objective of the two experiments and the full details of the experiments are in the following two chapters.

The Formula Editor Experiment

In the formula editor experiment, subjects are given the task of transcribing algebraic formulas into a linear string using two editors. One editor, the linear editor, operates on objects that are linear character strings like those in a text editor. The other editor, the semantic editor, operates on structured objects of <operant><operator> <operant> which reflects the object structure of the formula. The text editor will cause users to adopt many extra subgoals to map from the formula structure to the string structure. These extra subgoals are difficult to acquire, to verify, and they accumulate quickly causing memory overload leading to long performance time and errors. The semantic editor, as compared to the text editor, though it produces almost identical numbers of keystrokes in the final action sequence, poses few cognitive problems to the users as the subgoal structure reflects the task goal of recursively defining the operants. The subgoals in the semantic editor space do not accumulate like those in the text editor space as they are not recursive and are self-terminating. They do not contribute to the memory load.

This experiment illustrates why similar interfaces may produce very different subgoal structures thus affecting the performance time drastically. Also, it illustrates

tasks that demand excessive cognitive resources may not become skilled through practice. It also illustrates that an interface that allows an efficient execution affects the task acquisition time, thus violating the KLM model's assumption.

The Lotus Menu Experiment

In the Lotus menu traversal experiment, novice subjects learn how to execute simple spreadsheet tasks by issuing commands through the menu. The subjects see one of two versions of the menus and one of two versions of the instructions. The new menu structure as compared to the original Lotus menu has a consistent structure in terms of always having the spreadsheet objects at the first level and the possible operations on the objects at the second level of the menu. This consistent structure allows users to form a consistent subgoal structure of always looking for the object of interest first followed by the action of interest. The two versions of the instructions further constrain whether users can form this consistent subgoal structure by either describing the object first or the action first. We expect that when there is a mismatch between how the users think of the task (the instruction) and how the system orders the commands (the structure of the menu), an extra subgoals will need to be created to buffer the out of order command. These extra subgoals will again lengthen performance time and increase error rate.

The lotus menu experiment shows that an interface should not only be internally consistent, it should also be consistent with how users decompose the task goals into action subgoals. It illustrates that the task acquisition time and the task execution time are not independent as suggested by the KLM model's assumption. The interface that allows users to execute their task as conceived in the task space will result in better performance. The next two chapters describe these two experiments in detail.

CHAPTER 3

THE FORMULA EDITOR EXPERIMENT

In this chapter, we will use a formula entry task to illustrate the subgoal theory to show the importance of task-system match to eliminate extraneous subgoals. It will show how an editor that enters mathematical formula as a linear string forces users to break down the task of formula entry into a series of short and unnatural subgoals as the task space and system space operate on different objects. The time for cognitive activities overwhelms the keystroke time thus making the GOMS keystroke level model prediction inaccurate. We will then show how a simple redesign of the editor eliminates much of the cognitive bottleneck due to the difficulties in acquiring and evaluating the extraneous subgoals.

This chapter will start with the description of the formula entry task, the formula structure, and the cognitive processes involved in entering formulas linearly. A pilot study and its results are then presented to verify that users face difficulties when keying in formulas linearly. The chapter next presents a cognitive model that accounts for these difficulties. This chapter concludes by describing an improved editor that reduces some of these difficulties and an experiment and its results that compares the two editors' ease of use.

3.1   The Formula Entry Task Using a Linear Editor

The task of keying in formulas using a plain text editor is common in spreadsheet applications. Users need to translate the formula into a linear string as they key it in. For

58

example, the formula $(A+B)*\frac{H}{S-B}$ when keyed in as a linear string becomes

$(A+B)*(H/(S-B))$.

Formulas belong to a class of languages called functional languages which also includes LISP statements, PostScripts language, etc. In a functional language, parentheses are used to delimit the semantic units in a statement consisting of embedded functional statements. The high cognitive demand when keying in such language statements on computers using a linear text editor is a consequence of having to keep track of the multilevel embedded statements and parentheses. Users need to employ extra subgoals to translate the hierarchical structure of the functional language statement into a linear string. Users need to plan for open parentheses, remember where to close the parentheses, and parse the statement to keep track of the level that they are in. Complex statements often lead to overextended cognitive resources, long pauses, and errors when keyed in.

Keying formulas is chosen as the task for cognitive modeling in our experiment as it is a common task, especially for spreadsheet users and programmers. Analyzing the cognitive difficulties in using a functional language is important since past research have revealed that there are many errors in spreadsheet models' formulas [Brown & Gould 1987; Ditlea 1987] and in LISP statements [Anderson & Jeffries, 1985]. Since formulas are representative of functional languages, the discussions in this chapter can be generalized to other functional languages. Although there are editors that display formulas in a typeset format on graphics workstations, most editors still require users to transcribe formulas into a linear format when keying in and display the keyed in formulas in a linear format. Also, linear editors are the standard tools for editing other functional languages. We thus choose to analyze the cognitive process of using such a linear editor interface when keying in formulas. Once we understand the cognitive process, we can then design a better interface for keying in formulas or functional language statements.

We choose to analyze only formulas with simple binary operators: plus, minus, division, multiplication and power. We ignore more complex operators like integration, summation and differentiation to keep the formulas simple. This will not lead to any loss of generality or power for our discussion as the purpose of the research is not to design a more powerful formula editor but to illustrate the difficulties of managing the extra subgoals when there is a task-system mismatch. Formulas with simple operators will be sufficient to illustrate the cognitive difficulties of the formula entry task.

To understand the cognitive difficulties of formula entry, we first need to understand the structure of formulas. The next section presents a formal analysis of the structure and meaning of formulas.

3.2   The Structure and Meaning of Formulas

We need to understand the structure and meaning of formulas to understand the difficulties users have when they key in formulas before we can model the underlying cognitive process. Although a formula is just a string of symbols when keyed in, it has an underlying structure. As Figure 3.1 below shows, a formula is built up recursively from an operator joining two components. Each component is an operant and the operant itself can be a formula, i.e., another operator joining two other operants.

In Figure 3.1, <O> (as in object) is used for operant and <A> (as in action) is used for operator. Each of these <O><A><O> expressions in the task space constitutes a "*semantic unit*" in a formula as it can be a meaningful constituent of a higher level formula. Users can think of the <O><A><O> sequence as an action upon two objects. The parentheses around the <O><A><O> units make them into a higher level operant. We will call such operants with parentheses around them *complex operants*.

Parentheses are used in formulas to delimit component boundaries. They are essential in a linearly displayed formula to delimit the semantic units. They are also used

to change the precedence of operators by stating the semantic units explicitly. For example, 4*(2+3) and 4*2+3 yield different results. The parentheses around 2+3 in the first example made it into a semantic unit and it has to be evaluated first. Consequently, a pair of parentheses is a surface structure that indicates the formula within the pair of parentheses is at a lower level and should be treated as an operant for the higher level formula.

Figure 3.1   The Recursive Structure of a Formula
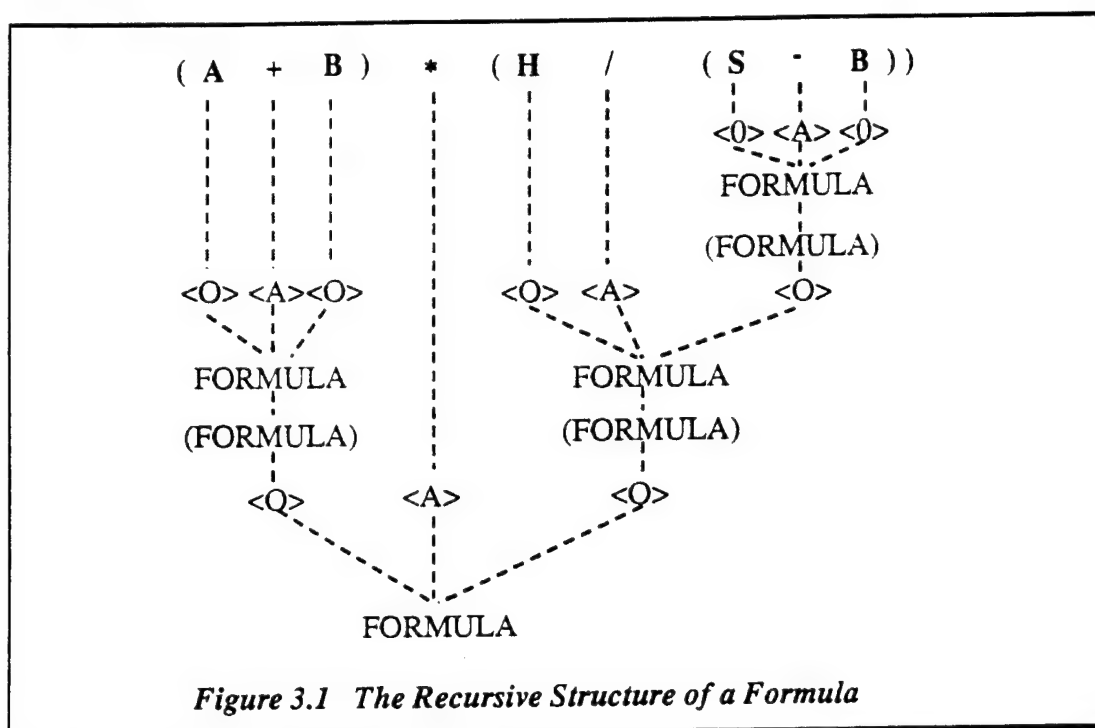
Based on the recursive structure of formulas in Figure 3.1, the meaning and structure of formulas is succinctly defined in Figure 3.2 as a formal grammar in BNF notation. We again see from the grammar that parentheses are symbols that make a formula into an operant. The parentheses mark the boundary of a 'semantically' complete unit of formula and make it into an operant. A formula is thus built up recursively with nested formulas.

It is this nested structure of formulas that create cognitive difficulties when users face the task of either keying a formula linearly or parsing a linear formula visually to

understand its structure. The next section discusses what these difficulties are and users' strategies when keying formulas to alleviate the difficulties.

```
<Formula>    :=   <Operant><Operator><Operant>

<Operant>    :=   < (Formula) >|< Constant >|< Variable >

<Operator>   :=   + | - | * | ^ | /
```

Figure 3.2 A Formal Grammar for the Meaning of a Formula

3.3   The Cognitive Process of the Formula Entry Task

3.3.1   The Cognitive Resource Limitation

The embedded structure of formulas creates cognitive difficulties for users when keying a complex formula in a linearly. To enter a long and complex formula linearly and not exceed his cognitive resources, the user needs to break the goal of entering the formula into subgoals of entering one portion of the formula at a time so that each subgoal can be performed within working memory limits. However, the user needs extra subgoals to translate the <O><A><O> semantic structure in the task space to the linear character string structure in the system space. These extra subgoals arise as the user cannot think in terms of the object structure in the task space. Extra subgoals in the form of opening and closing parentheses are needed to bridge the semantic gaps. The cognitive difficulties arise from the planning, keeping track of, and evaluation for completion of such subgoals that do not reflect the task structure.

When keying in highly nested formulas, the cognitive resource limitation stems from the extraneous subgoals that result in excessive working memory load (a) to plan for the opening parentheses needed; (b) to keep track of nested subgoals of completing a

portion of the formula by balancing parentheses; and (c) to parse the linear string to mentally translate it back into its hierarchical nested structure to evaluate whether the subgoals are successfully completed. Looking at such a linear presentation of multilevel formula creates both high perceptual load and working memory load. It is demanding to parse the nested semantic units displayed linearly on the screen as it requires lots of working memory capacity to reconstruct mentally the semantic units by tracking the corresponding pairs of parentheses visually.

3.3.2 The Different Strategies to Enter Formulas

To alleviate such resource limitations, users may adopt different strategies to key in formulas linearly. Such different strategies will not totally alleviate the cognitive difficulties but will create different tradeoffs among the needs for planning, tracking, and evaluation of subgoals. Different strategies will impose different subgoal structures to break the task into different sequences of subtasks. There are three strategies users could adopt: top-down, bottom-up or left-to-right. Figure 3.3 depicts the steps involved in creating a simple formula using different strategies.

The Top-Down and Bottom-Up Strategies. Using these two strategies, users set subgoals of entering complete semantic units (O-A-O) before proceeding to the next level of embedded formula. Figure 3.3 shows that there will be lots of navigation within the formula using the left and right cursors keys when using these two strategies. In a top-down approach, users start at the top level keying in the top most level O-A-O and expand complex operants one level at a time to the lowest level. Each complex operant is created by first creating a pair of parentheses and then filling in the formula between the parentheses when the user is at the next level of expansion. It is just the opposite in the bottom-up approach. Users start at the deepest level, key in the deepest level O-A-O and

then proceed to the next level up by placing a pair of parentheses around the lower level

O-A-O semantic units making them into complex operants.

The formula, $(A+B)*\dfrac{H}{S-B}$ in Figure 3.1, is used to illustrate the three different strategies.

A **top-down** approach:

Current Screen display	Next key(s) pressed
	()*()
()*()	← ← ← ← A+B
(A+B)*()	→ → → H/()
(A+B)*(H/())	← S-B
(A+B)*(H/(S-B))	

A **bottom-up** approach:

Current Screen display	Next key(s) pressed
	S-B
S-B	← ← ← (→ → →) ← ← ← ← H/
H/(S-B)	← ← A+B
A+B H/(S-B)	← ← ← (→ → →) → (→ → → → → → →)
(A+B)*(H/(S-B))	

A **left-to-right** approach

(A+B)*(	remember one parenthesis to close and position
(A+B)*(H/(	remember another parenthesis to close
(A+B)*(H/(S-B)	close second parenthesis
(A+B)*(H/(S-B))	close first parenthesis

Figure 3.3 **The Three Strategies for Keying Linear Formulas**

These two strategies attempt to operate on the semantic structure of <O><A><O> despite the linear structure of the task space. These strategies will ensure that there are no unbalanced parentheses but will require many cursor movements to get to the desired locations to insert parentheses around formula or to insert formulas within a pair of

parentheses. Moving within the formulas not only creates extra cursor movements but also will impose high memory and perceptual load. By navigating within the formula, it is difficult for the user to keep track of the formula level the user is at and has to constantly parse the partially completed linear formula mentally into a hierarchical structure to insert the nested formulas or the parentheses at the right places. Users may not use these two strategies due to the excessive jumping around causing high perceptual and memory load. However, the main reason that users may not adopt these two strategies is that the operators and objects in the system space (the linear editor) color the way users think of the task. The left-to-right nature of the editor induces users to tackle the task of formula entry in a left-to-right fashion.

The Left-to-right Strategy. In this strategy, users try to proceed in a strictly left-to-right fashion by translating the nested hierarchical structure of the formula into a linear string as the formula is keyed in. As the users key in the formula, they will need to plan for extra subgoals to see where to open parentheses, remember where and when to close parentheses, and to evaluate whether the linear formula is correctly constructed. Thus the subgoals in this strategy are not so much creating O-A-O sequences but are opening and balancing parentheses.

This strategy is economical on keystrokes because it does not require moving around but imposes a high memory load to keep track of unbalanced parentheses. Also, there is a high cognitive load to translate the hierarchical formula into a linear string on the fly. If a formula is highly embedded, it will be impossible to keep track of all the uncompleted subgoals of closing parentheses because simultaneously, the working memory is being used to parse the formula into a linear string to know the right places for the open and closing parentheses. Due to this high memory load, we postulate that users will forget to open and close parentheses. However, if while entering the formula the user feels that the formula is not balanced, the user can rescan the formula to check

whether closing parenthesis is needed or backtrack to insert forgotten opening parenthesis. The checking for unbalanced parentheses will take a long time because the user again needs to translate the linear string into a hierarchical structure mentally.

The left-to-right strategy is demanding in terms of the cognitive resources but users may still prefer this strategy as this is how a plain editor is used in normal text processing situations. Text is usually entered in a left-to-right fashion. Users may be constrained by this inner system space and proceed in a left-to-right fashion when using the plain text editor to enter the formulas.

Besides using one of the above three strategies, users can use a hybrid of the above three strategies as none of the strategies is ideal. There are tradeoffs among the number of extra keystrokes, memory load, and the need to check for the correctness for the different strategies for keying formulas. We use the following pilot study to investigate what strategies subjects generally adopt when they are required to enter a formula using a linear editor. It also will validate the claim that users will face difficulties when keying in formulas linearly. If subjects adopt the same strategy, we can then model the cognitive difficulties using a cognitive model and see whether it could predict the cognitive difficulties revealed through error positions and long pauses in the keystroke pattern.

3.4   The Pilot Study

The Objective

The purpose of the experiment is to understand the cognitive process involved in keying in formulas in a linear format. The outcome of the experiment will reveal which of the three strategies subjects adopt. The keystroke timing and error data should help us understand the difficulties users face when using a linear editor to key in formulas. A

cognitive model of the formula keying process can then be developed and a new formula editor will then be designed to address specifically the difficulties revealed in the model.

The Task·

Subjects were required to use a linear editor to directly key in formulas presented one at a time on 3x5 cards. Numbers instead of alphabetical names were used as operants in the formulas to minimize the effect of different typing speed. Figure 3.4 presents two examples of such formulas. We chose a transcription task here rather than a formula composition task (where word problems are presented and subjects are required to key in the resulting formulas) because the composition task would add problem reading time and problem solving time to the time due purely to subgoal formulation, tracking, and evaluation.

The level of complexity of the formulas used in the experiment range from two to four levels of embedded parentheses. The range of complexity is reasonable and representative of functional languages. In other functional language like LISP, there can be an even higher level of embedding. Examples of two level and four level embedded formulas are $((2+4)*3)/4$ and $((((1+2)*3)\char94 4)/5)/(2+3)$ respectively.

$$\frac{2+3*\left\{\left(\frac{2+3}{4}\right)^5+2\right\}}{2}$$

$$\frac{2*\left\{\sqrt{1+2}+\left(\frac{3}{4+5}\right)\right\}}{1+(2*3)^2}$$

Figure 3.4   Examples of Formulas Used in Experiment

To encourage users to employ different strategies in entering formulas, four different cursor movements keys were provided. Besides the usual single step left and

right arrow keys, we provide <eol> (end-of-line) and <sol> (start-of-line) to help users move quickly within the formula. These keys provide an easy way for users to adopt the top-down or the bottom-up approach instead of a left-to-right approach in keying in formulas.

The Formula Chunking Task

At the end of the experiment, subjects were also asked to circle chunks in formulas that they felt were complete semantic units. They did this task for 10 formulas on a sheet similar in complexity to those used in the experiment. Five of the formulas were presented in the same typeset format as those on the cards and the remaining five were presented linearly as displayed on the screen. The formulas presented alternate between the two formats.

The Equipment

An IBM PC was set up with an IBM's keysave program running in the background of a word processor (Final Word II) to collect keystroke timing. The keysave program has a resolution of 17 milliseconds. Since we are interested in the cognitive process but not the typing speed of subjects, we had to minimize the effects due to different typing speed and familiarity with keyboard. Consequently, we programmed the keyboard such that subjects only needed to use the keypad portion of the keyboard and with only one hand. This avoided the problem of subjects' spending time in searching for seldom-used keys, like the operators and parentheses. Also, only the numbers 1 to 6 were used as variables to avoid typing alphabetical variable names. Figure 3.5 depicts the keyboard mapping. Labeled color-key stickers were placed over the keypad to show the key mapping. The stickers were color-coded to denote different functions of the keys, i.e., operators, operants and cursor movement keys. In addition, to minimize distractions,

we programmed the word processor to show only one line of text on the screen. Every formula used in the experiment fit on a single line when keyed in.

Figure 3.5 Keyboard for Formula Entry Using the Linear Editor
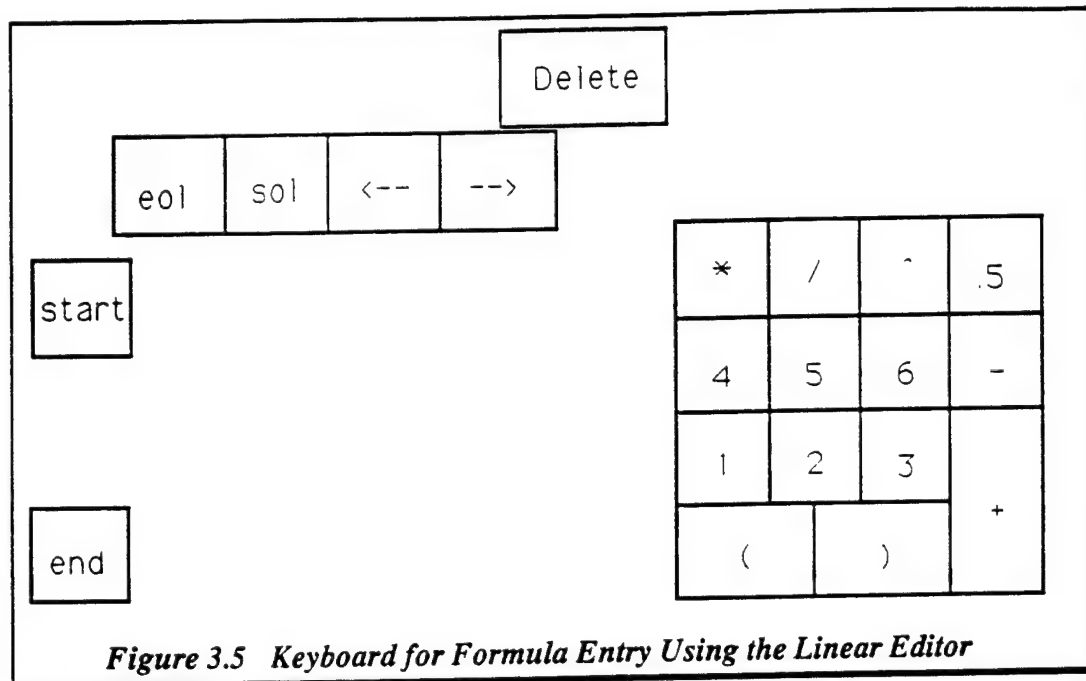
The Subjects

The subjects were five PhD students and two faculty members working in the Human-Computer Interaction Laboratory at the University of Michigan. Since all the subjects have advanced degrees in scientific fields, it is reasonable to assume that they were familiar with formulas.

Instructions

The experimenter read the instructions (see Appendix A), gave a tour of the keyboard to the subjects and answered questions subjects had. Subjects were told to proceed as accurately and as quickly as possible but they were not told to use any specific strategy when they typed in the formulas.

Procedure

The whole experiment took about 40 minutes. Subjects practiced with a set of ten warm-up formulas to become familiar with the experimental procedure. In the actual data collection phase, subjects had to key in 10 formulas presented one at a time on 3x5 cards. Appendix B gives a complete listing of the 10 practice and 10 actual formulas used in the experiments. After the session, subjects performed the formula chunking task on 10 different formulas. To balance the order-effect, half the subjects keyed in the formulas in one order, and the other half did it in the reverse order. The formulas with different complexity were randomly distributed.

Subjects had to press the <start> key before looking at a card and had to press the <end> key when they were done keying in the formula. The <start> and <end> key provided a time stamp for analyzing the task acquisition time and checking time for each formula. The experimenter left the subjects alone to key in the formulas after the practice trials in order not to create any demand effects. The keystroke time stamp file and the file that contained the completed formulas allowed us to analyze both the errors committed and the time required to key in each formula.

3.5   Results of the Pilot Study

We first present the results of the formula chunking task and the strategy adopted by users to key in formulas. The error rates and keystroke timing are then analyzed to help us develop a cognitive model for the process of keying formulas linearly. The errors and keystroke times are not analyzed in detail but are only used to help us understand the difficulties. Detailed analysis is performed on the results of the actual experiment proposed later in this chapter.

The Formula Chunking Task

Subjects did not circle all the semantic units in the formulas. They did circle as many units as they could before it got too cluttered (Appendix C gives an example of a subject's circled formula sheet). The chunks left out were random and did not correlate with the types of errors when entering the formulas using the linear editor. No subjects made even a single error in chunking semantic units in the 10 formulas regardless of the format presented to them. They recognized that formulas are made up of formulas (complex operants) recursively. The results showed that subjects do not treat formulas, even when presented linearly, just as a string of symbols, but understand the underlying structure of formulas.

Strategy Adopted When Keying Formulas

The keystrokes captured revealed that all subjects adopted a left-to-right strategy in keying the formula. Subjects would mentally parse the formula into a linear format as they keyed in the formula to avoid using cursor keys as much as possible. Although cursor movement keys were provided, no subjects employed a top-down or a bottom-up strategy. Subjects backtracked only when they realized that they had forgotten to insert an open parenthesis. Such backtracking was not preplanned as it would be in a bottom-up approach; it appeared to serve as error recovery since there were long pauses before the backtracking.

Although the grammar for the formula depicts that working with a top-down or bottom-up strategy is more efficient, subjects were not willing or able to do so. They were artificially constrained by the linear editor system space imposed on them. They did not break the formula entry tasks into subgoals of completing semantic units but instead used subgoals of opening and closing parentheses to bridge the task-system space mismatch so that they could proceed in a left-to-right fashion.

The left-to-right strategy will overextend the working memory for highly nested formulas as there will be too many subgoals of closing parentheses pending. Also, these subgoals do not match the task goal structure of recursively creating O-A-O structures. The subgoals are difficult to acquire, monitor and validate. We expected to see many errors and long pauses for these formulas.

Error Rate and Types of Error

There were errors in 30 of the 70 formulas that the seven subjects (10 for each subjects) keyed in. The high error rates reflects that keying formulas linearly is a cognitively demanding task. Among the erroneous formulas, 37% had multiple errors in them. Of all the errors, 94% involve missing parentheses or misplaced parentheses, 16% of the errors involve semantic errors where parentheses are balanced but are misplaced in a way that completely changes the meaning of the formulas. These semantic errors reflect the perceptual load problem and memory load problem of parsing out the semantic units in a linearly presented formula. Users can count parentheses to ensure that they are balanced but syntactically correct formulas are not necessarily semantically correct. There are almost equal numbers of errors involving opening and closing parentheses. This indicates that balancing parentheses is not the main problem here but rather translating the hierarchical structure to a linear structure.

Error Positions

In the results, the positions of the errors where parentheses were either missing or misplaced revealed that memory load was a problem. Generally, when there were more levels of embedded formulas within a pair of parentheses, it was more likely that the opening or (and) the closing parenthesis was misplaced or missing. Subjects had difficulties in parsing the formulas to plan for subgoals to open parentheses and in remembering the subgoals to balance parentheses.

Figure 3.6 depicts the positions and the number of errors at these positions in a bar chart. When there was only one embedded formula, e.g., (3+5), subjects made no errors in keying the parentheses. When there were four levels of embedded formulas, e.g., (1*(2^(5*(2/(3-2))))), subjects made a total of 19 errors keying the parentheses.

The Keystroke Data

We expect that the keystroke time will be short since subjects made so many mistakes. In fact, there were many long pauses over 5 seconds. This implies that subjects had difficulty ensuring that the formulas were correctly keyed in despite spending a long time planning for open parentheses and checking for correctness of closing parentheses. The difficulty arises from the need to use extra subgoals to translate from the <O><A><O> formula structure to the linear string structure.

The keystroke time pattern will reveal the subgoal structure adopted by the users. We expect long pauses to occur before and after each subgoal for subgoal acquisition and evaluation. The Keystroke Level Model's parameters should help us interpret our data. We learned from the keystroke level model parameters that the time (k) for typing a single character range from .08 seconds (fastest typist) to 1.20 seconds (unfamiliar with keyboard). The mental preparation time (M) is about 1.35 seconds. Since subjects only need to use a small keypad, we do not k to exceed 0.5 seconds. Consequently, the keystroke level model parameters cannot account for the keystrokes that take more than about two seconds. We will use this time as a guide and imply that any longer keystroke time will include either task acquisition or evaluation, activities that involve demanding cognitive activities like scanning and parsing the formula.

A subgoal leading to a subtask is then taken as a sequence of keystrokes that contain only M or k without any other cognitive activities. The subtask here is equivalent to the unit task in the GOMS model. Once there is cognitive activity other than mental

preparation (M), the users have created a new subgoal to keep the subtask performance within their cognitive resource limitation. During a subgoal for keying formulas, the users look at the formula presented on the card, compose and remember the next few keystrokes to be keyed in that can be kept in working memory, then key in a burst as reflected by the pure keystroke time. The subgoals formulated will help us understand the difficulties users face when keying in formulas.

Figure 3.6   The Number of Errors at Parenthesis' Locations with Different Levels of Embedded Formulas.
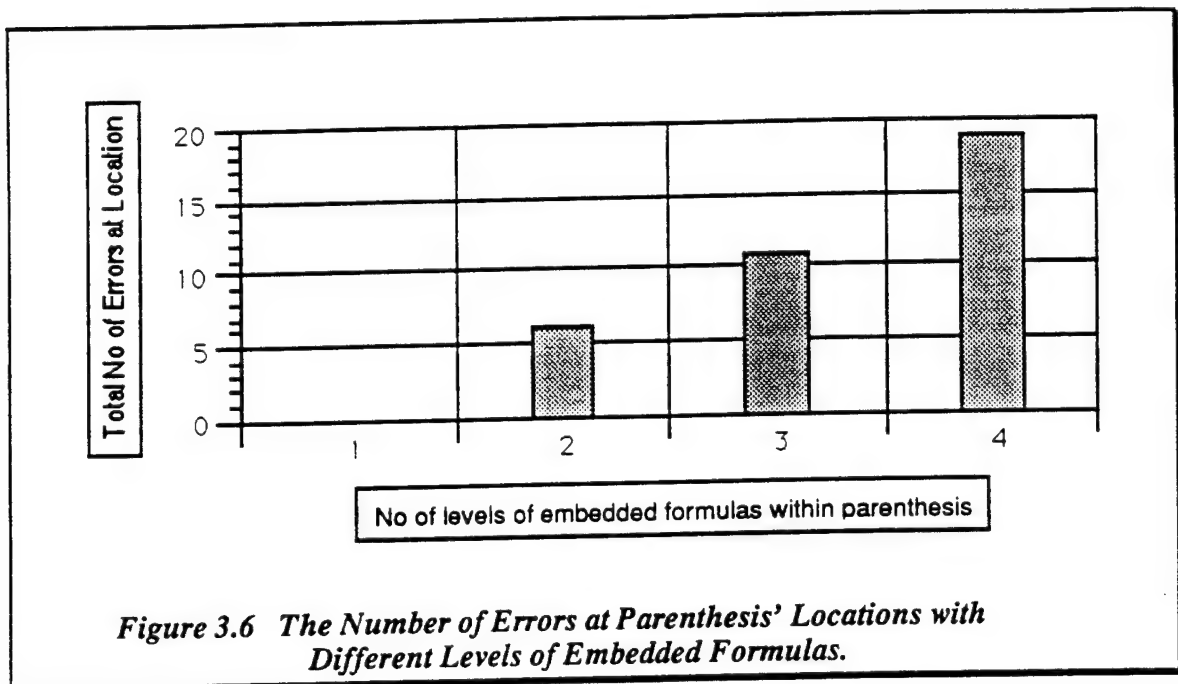
Cognitive activities occur at the boundaries of subgoals. Before executing such subtasks, there is the *task acquisition*; and after that, there is *task evaluation*. Task acquisition time can be long when the users face a complex operant. They have to decide whether a parenthesis is needed and if so, they have to parse the formula on the card then decide and remember where the closing parenthesis should go. Such pauses are evident in our data when there is an open parenthesis.

Task evaluation time is reflected as long pauses after the subtask. Such pauses are unusually long when the users cannot remember where to insert the close parenthesis and have to parse the displayed formula to check for correctness. The checking time is as

high as 24.22 seconds in our results. This time is much higher than the scan time parameter of 2.87 seconds Olson and Nilsen [1988] found in their formula keying task. Their scan time is for locating the coordinates of the variables; whereas our scan time is to parse for the semantic units of a formula. The latter is a much more cognitively demanding task as it often exceeds the working memory limit.

The keystroke data for the 10 formulas that the subjects keyed in exhibits the same irregular pattern of long pauses. The time for individual keystroke range from a low of 0.22 seconds to a high of 24.22 seconds, a factor of more than 100. The locations of the long pauses are generally those where an open parenthesis is needed to start a complex operant and where there is a closing parenthesis needed to complete a complex operant. However, subjects do not always plan before keying. Some prefer to plan for a long time then press the keystroke; some prefer to quickly key in the keystroke then spend a long time in checking. Thus, the pause does not increase with the level of embedding. The average time to open a parenthesis is 3.57 seconds and the average time to close a parenthesis is 3.77 seconds. The average keystroke time for an operator is 1.39 seconds and is 1.43 seconds for a variable. Subjects spent more time in opening and closing parentheses than typing variables or operators. Since the times to key parentheses are more than two seconds, they have to include planning time and evaluation time.

These frequent long pauses indicate that when faced with cognitive difficulties of parsing and scanning to translate the different objects manipulated in the task and system space, subjects were forced to adopt subgoals that had only a single keystroke. Such subgoal of opening or closing parenthesis is short, unnatural, and does not confirm to the <O><A><O> structure for formula entry.

The only time the subjects could adopt subgoals that are meaningful was when they were keying in sequence of <O><A><O> with both operant being simple, e.g., 1+2, 4+5 and 2*3. These clusters of three keystrokes are obvious subgoals not only as

reflected by the keystroke timing exhibited in the results, but also as defined by the grammar and intuitive reasoning. They are short chunks of three elements that can be kept in working memory; they only have simple operants that need no cognitive activities for checking and planning; and they each form a complete semantic unit in the formula.

Discussion of Pilot Study Results

The long pauses in the keystroke data overwhelm the keystroke time thus making prediction of a keystroke-level model inaccurate and unusable for keying formulas linearly. These long pauses, caused by the difficulties in acquiring and verifying the extra subgoals, are not modeled by the skill-based and rule-based GOMS model.

The error pattern fits with the findings from a series of experiments conducted by Anderson and Jeffries [1985] to investigate novice LISP errors. In their experiments, subjects evaluated highly embedded LISP statements. They found that subjects' errors seemed to result from slips rather than misconception. Our results from the chunking task revealed our subjects also did not have misconception of formulas. They attributed the slips to loss of information in working memory when parsing the LISP statements. We also attribute errors to working memory overload due to parsing and remembering. Also, they found that in writing LISP expressions, most mistakes involved errors in parenthesization and that these errors increased with the level of embedding. Most of the parenthesis errors produced semantic errors. Our results reflect the same pattern but we have fewer semantic errors in comparison. The reason we have more syntactic errors could be that the parentheses in our single character variable formula are more cluttered compared to those in LISP statements. When there are too many opening or closing parentheses cluttered together, it is very difficult to visually parse the formula into a hierarchical structure and check for its correctness.

The error pattern and keystroke timing obtained in the pilot study confirm our hypothesis that keying formula linearly is a cognitively demanding task. We will next model the cognitive processes and difficulties people have when keying formulas linearly.

3.6   A Cognitive Process Model for Keying Formulas Linearly

We develop a cognitive model that formally describes the left-to-right strategy of the users and succinctly predicts the cognitive bottleneck where there are likely to be long pauses or errors. The model also will reveal the unnatural subgoals adopted by the users. The model, depicted in Figure 3.7, is written in NGOMSL notation [Kieras, 1988], an English like description of a GOMS model. Figure 3.8 depicts the same model in a schematic diagram. Note that in Figure 3.8 there are many cognitive operators (those in round-corner rectangles) that take a long time to perform and are error prone.

In the model, the user starts with a top level goal of entering the formula (line A in Figure 3.7). The user then proceeds in a left-to-right fashion and tries to type in the two operants (A1 and A3). If the operant is complex, he invokes Method C to enter the operant by scanning the formula to see where the closing parenthesis is needed (D1), retain that position in memory (D2) and retain that a closing parenthesis is needed (D3). The user then proceeds to invoke another goal to enter the embedded formula within the complex operant (C2). This is where the recursive structure of the formulas will cause problems for the users as there will be many uncompleted subgoals stacking up in the working memory. This will exceed the users' working memory limit and users will have to rescan the formula to recover their position (G). Users will need to rescan the linear formula to check whether their subgoals have been accomplished. Due to memory load, users almost always forget their uncompleted subgoals but will constantly rescan the formula to reconstruct the goal sequence.

Cognitive Model for Linear Editor Formula Entry Task

A. Method to accomplish goal of entering formula
1. Accomplish the goal of entering operant
2. Type Operator
3. Accomplish the goal of entering operant
4. Validate that the formula is balanced and correct
5. Report Goal Accomplished

B. Selection rule for the goal of entering operant
1. If the operant is simple, then type the variable
2. If the operant is complex, then accomplish the goal of entering complex operant

C. Method to accomplish the goal of entering complex operant
1. Accomplish the goal of opening parenthesis
2. Accomplish the goal of entering formula
3. Accomplish the goal of closing parenthesis
4. Report goal accomplished

D. Method to accomplish the goal of opening parenthesis
1. Scan formula and decide where closing parenthesis is needed
2. Retain the position of closing parenthesis
3. Retain that a closing parenthesis is needed
4. Type an open parenthesis
5. Report goal accomplished

E. Selection rule for the goal of closing parenthesis
1. If memory not overloaded, accomplish goal of typing closed parenthesis
2. If memory overloaded but suspect unbalanced parentheses, accomplish goal of position recovery

F. Method to accomplish the goal of typing closed parenthesis
1. Type a closed parenthesis
2. Report goal accomplished

G. Method to accomplish the goal of position recovery
1. Rescan formula to see whether closing parenthesis needed
2. If yes, type a closed parenthesis
3. Report goal accomplished

Figure 3.7. A Cognitive Model for Using the Linear Editor

The model reveals that the subgoals users employ are small and semantically

meaningless. They are opening parenthesis, closing parenthesis, and recovering position.

Such unit tasks are often down to only one keystroke. This will make the unaltered keystroke level-model unhelpful since the subgoal acquisition time overwhelms the keystroke time.

Figure 3.8 A Schematic Diagram for Using the Linear Formula Editor
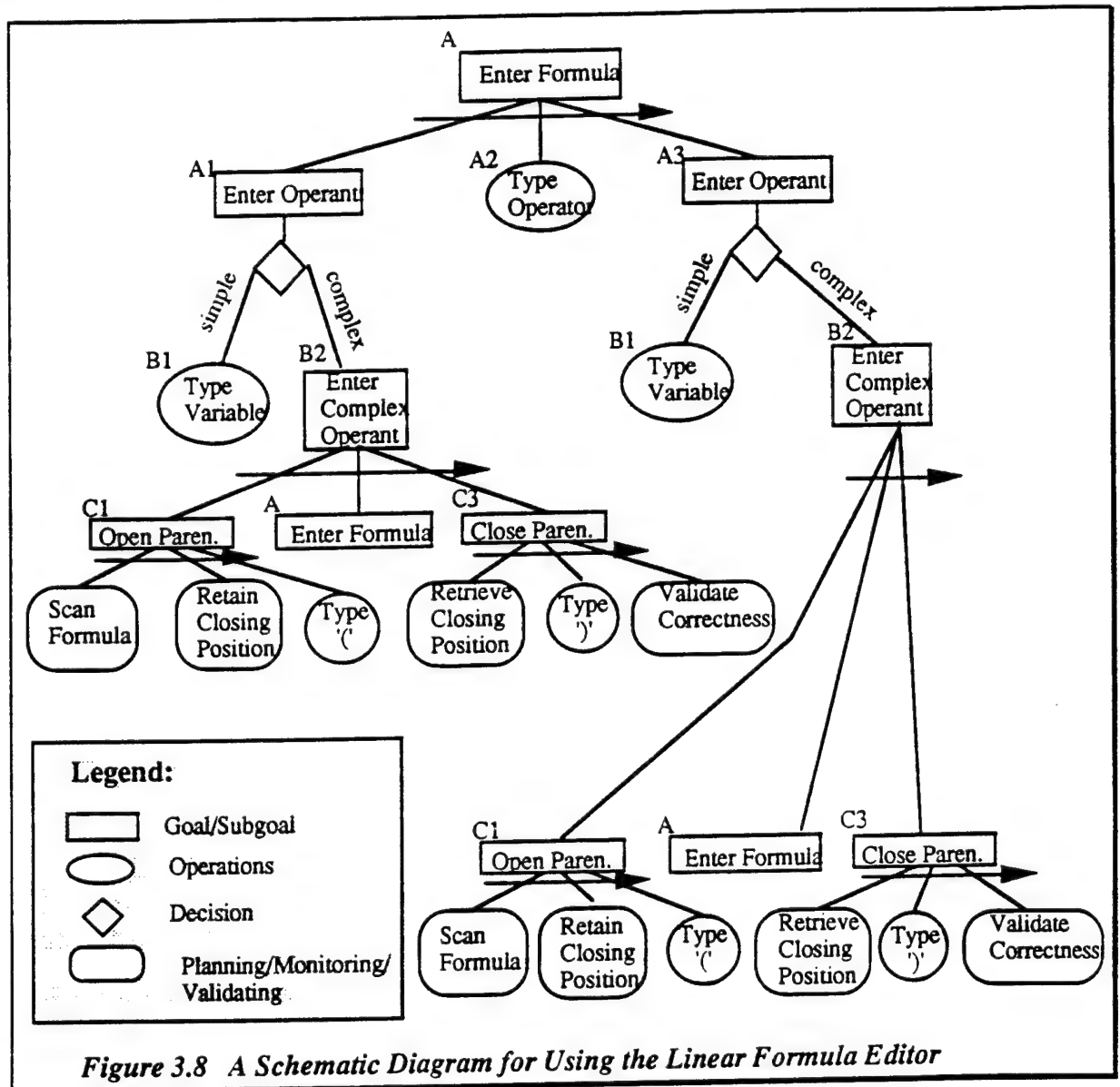
From the psychology literature, we know that there is a limited capacity to our working memory [Miller 1956; Waugh and Norman 1965]. In our cognitive model, working memory is used to hold the active productions, the uncompleted subgoals, and for mentally constructing the semantic units when visually parsing the formula. The

recursive nature of formulas loads the working memory rapidly with the subgoals of completing complex operants with right parentheses as the level of embedding increases. The working memory overload problem is accentuated since users also need working memory space to parse the highly embedded formulas keyed in to validate its correctness. When entering such formulas, the model predicts that users either will forget to close the right parenthesis due to memory overload, or will have to pause a long time to parse the linearly displayed formula to check for correctness. It follows that the error rate should increase with the level of embedding.

The cognitive model predicts that there will be a long pause when subjects check for the need for closing parentheses (G1). Checking, however, will not ensure all errors will be eliminated. As visually parsing a linear formula to construct mentally the semantic units is highly demanding on working memory, it might be impossible to reconstruct a highly embedded formula. Users will often resort to counting parentheses to check for equal numbers of right and left parentheses. This sometimes leads to syntactically balanced but semantically incorrect formulas. Furthermore, counting parentheses that are clustered together is visually demanding and can lead to syntactic errors of unbalanced parentheses.

The cognitive model also predicts that there will be a long pause when users face a complex operant. The pause is for scanning ahead to see where the closing parenthesis is needed and for loading this subgoal into memory. Such parsing again requires working memory and might displace previous subgoals of closing unbalanced parentheses.

Overall, there are many cognitive bottlenecks as predicted by the above systems. We now describe the design of a formula editor that specifically addresses these difficulties.

3.7   The Semantic Formula Editor

We design a formula editor that allows users to break the task down into sequences of subtasks conforming to the recursive semantic structure of formulas. The interface will allow users to manipulate the <O><A><O> structures of a formula. The interface will allow users to concentrate on the task of entering formulas but not the task of translating the formula into a linear structure and keeping track of extraneous subgoals. The interface will keep track of the semantic structure of formulas to avoid the need to construct mentally the semantic units. The editor is designed to reduce the need for planning and checking. We will call this editor the *'semantic editor'* as it manipulates and keeps track of the semantic structure of formulas.

From the formal grammar of formulas in Figure 3.2, we understand that formulas are built up recursively with the basic structure or the semantic units of <operant><operator><operant>. The design of the semantic editor evolves around this basic structure of the formula. It allows users to create complex formulas from the basic semantic units recursively without having to worry explicitly about parentheses. Parentheses are generated automatically as a byproduct when users create a complex operant. There is a special key to create complex operants and is denoted by "()" on the keyboard.

To compare directly the semantic editor with the linear editor on a formula transcription task, the semantic editor is designed for a left-to-right strategy for entering the components of a formula. This will turn out to be an in-order traversal of the syntax tree. Working with one complete semantic unit of <O><A><O> at a time, the formula is expanded recursively from a left-to-right order as prescribed by the grammar. To ensure we have a direct comparison with the linear editor, the output of the semantic editor is made linear although we have versions that will produce a structured display of the formulas as they are keyed in.

To key in a formula, users employ sequences of three keystrokes of <O><A><O>. When the <operant> is simple, users just type the variable (a number in this case); when the <operant> is complex, users just press the '()' key. The '()' key is a special key for creating complex operants. These three keystrokes form a natural subgoal as it conforms to the semantic unit in formulas.

---

1. to key in **(1+2)*3**

current screen display	next key(s) pressed
blank	1 + 2
1+2_	()
1+2_	*
(1+2)*_	3
(1+2)*3_	

2. to key in **3*(1+2)**

current screen display	next key(s) pressed
blank	3 *
3*_	()
3*(_)	1+2
3*(1+2_)	

Figure 3.9   Two Examples of How the Semantic Editor Works

A slight complication arises when there is a sequence of consecutive simple operants, e.g., 1+2+3. Grammatically, it can be view as (1+2)+3 or 1+(2+3). Users can still use the complex operant key to maintain the sequences of three keystrokes. The editor can be programmed to check for operator precedence and not display the extra parentheses. However, in order not to present users with too much novelty, we chose not to enforce the binary nature of formulas but to allow users to type in the string as it is with 5 keystrokes like when using a linear editor. Figure 3.9 and Figure 3.10 give

examples of how the semantic editor is used to enter formulas to clarify the workings of the semantic editor.

In Figure 3.9 and 3.10, '_' denotes the current cursor position and '()' is the key to create complex operant. The bold print denotes highlighting on the screen. The sequence of keys pressed shown in the right column always falls into groups of three's as the key press sequence conforms to the <O><A><O> structure. There are two ways of creating complex operant depending on whether the complex operant is before or after an operator. In the first example, the complex operant is before the operator * and the complex operant is created after the embedded formula 1+2 is created. When the complex operant to be created is before an operator, pressing the '()' key before the operator key will highlight the next semantic chunk of the formula that can be made into a complex operant. Pressing any operator key after some semantic unit has been highlighted will insert a pair of parentheses around that unit before the operator and leave the cursor after the operator. In the second example, the complex operant is after the operator '*' and the complex operant is created before the embedded formula 1+2 is inserted. Pressing the '()' key after any operator will create a pair of parentheses after that operator and leave the cursor key within the newly created parentheses to insert the embedded formula. A slightly more complex example in Figure 3.10 will illustrate additional functions of the semantic editor.

In the example in Figure 3.10, notice that after keying .5+3, () was pressed twice to create the desired complex operant. Whenever () is pressed to create a complex operant before an operator, the smallest whole semantic chunk to the left of the cursor is highlighted. If the semantic chunk is not the one desired, pressing () again will highlight the next larger meaningful chunk. When the desired chunk is highlighted, pressing any operator key will create a pair of parentheses around the complex operant before the operator. However, it is infrequent that users need to check whether the correct semantic

chunk is highlighted. Normally, the first semantic chunk highlighted is the one desired: rarely, users need another one or two more '()' key presses to highlight the desired chunk. If a bigger semantic chunk is needed, the highlighting will help users parse the semantic structure of the formula and reduce errors and long pauses due to perceptual and memory load. Also, since only meaningful semantic units are highlighted, errors due to illegal syntax will be eliminated.

3. to key in $\dfrac{3*(\sqrt{1+2}+3)}{5}$

current screen display	next key pressed
3*_	()
3*(_)	1+2
3*(1+2_)	()
3*(**1+2**_)	^
3*((1+2)^_)	.5+3
3*((1+2)^.5+3_)	()
3*((**1+2**)**^.5+3**_)	()
3*((1+2)^.5+3)_	/
(3*((1+2)^.5+3)) /_	5
(3*((1+2)^.5+3)) / 5_	

Figure 3.10 Another Example of How the Semantic Editor Works

Since the editor keeps track of the underlying semantical structure of the formulas and users input formula according to the semantics of the formulas, the editor can detect any syntactic errors users make as they key in the formulas. Consequently, there can be no syntax errors when using the editor; errors will only be typographical errors or semantic errors. The latter can be caused either by a misunderstanding of the semantic structure of a formula or by not knowing how the semantic editor functions.

Error Corrections When Using the Semantic Editor

Since the editor works with semantic units of formulas, any error correction also will be based on semantic units. For simple errors like typographic errors, users can use the normal backspace key and retype the variable or operator. For correcting structural errors, there is a special 'UNDO' key. The UNDO key performs a few functions that reverse the effects of actions related to creating complex operants. If the UNDO key is used after some complex operant is highlighted but before the '()' key is pressed, the highlighting will be undone and cursor position restored to where it was just before highlighting. Pressing the UNDO key immediately after a complex operant is created will delete the pair of parentheses just created. The editor does not allow deletion of a single parenthesis as it will then not be able to track the semantic structure of the formula.

Structural Display of the Formula while it is Keyed in

As the editor keeps track of the underlying semantic structure of the formula, an important feature of the semantic editor is its ability to display the formula in a way that reflects its semantic structure while the formula is being keyed in. Different versions of the editor have been modified to display instantly while the formula is being keyed displaying (a) different types of parentheses to indicate different levels of parentheses, (b) numbers at the bottom of parentheses to indicate the levels of embedding, and (c) the formula on multiple lines with the depth corresponding to the level of embedding of the operant. Ultimately, the editor can display the formula in a typeset format on a graphics terminal while it is keyed in. Displaying formulas in a typeset format reduces the need to use parentheses to demarcate the semantic units. This will reduce the clutter on the screen. These improved displays will help users parse the formula and reduce the semantic gap on the output side.

This structural display capability is stripped from the editor used in the experiment to keep the differences between the two editors to the difficulties of translating external task to internal unit tasks. We next develop a cognitive model for using the semantic editor and then describe an experiment to compare users' performance when using the two editors.

3.8 A Cognitive Process Model for Using the Semantic Editor

Like before, we will use a cognitive model to model the cognitive process of using the semantic editor. Figure 3.11 presents the cognitive model in NGOMSYL notation and Figure 3.12 presents the same model in a schematic diagram. By using the semantic editor, the users' action of creating formula now conforms with the structure of the formula. There are only two types of subgoals corresponding to the two methods in the model. The crucial part is that each of these methods has an <O><A><O> structure. The users use the basic building block of <operant><operator><operant> recursively to build up the formula. When the operant is simple, the user just types the variable; when the operant is complex, the user just presses the complex operant key '()'. Parentheses are now a byproduct of the meaning of the formula as intended by the user; whereas in the linear text editor, parentheses are used explicitly to give meaning to formulas. Also, one very important difference between the two production systems is that the subgoals for the semantic editors are all self-closing; that is, they do not generate unfinished subgoals that need to be attended to later, e.g., closing parenthesis.

If we compare the cognitive model above with the cognitive model in Figure 3.7 that models the cognitive process of using a linear formula editor, we can see that the cognitive difficulties for entering formulas have been greatly reduced by the semantic editor. Although the two models produce about the same number of keystrokes when fully decomposed into keystroke-level operators, they have very different subgoal

structures. The subgoal structure when using the semantic editor reflects the task goal of creating the semantic units. When using the linear editor, the subgoals produced contain many single keystroke subtasks of opening and closing parenthesis. These subgoals are extraneous to the task of formula entry and require a long time to plan and track. There are no cognitive operators in Figure 3.12 such as scanning, checking, remembering, and recalling. The cognitive difficulties have been reduced in three areas: (a) no need to parse the formula to be keyed in to see where the closing parenthesis is needed; (b) no need to remember to close parentheses and thus preventing subgoals from accumulating; and (c) no need to visually parse the linear formula on screen to see if parentheses are balanced. These improvements reduce working memory and perceptual load and should help reduce errors and long pauses.

Cognitive Model for Using the Semantic Editor

A. Method to accomplish goal of entering formula
1. Accomplish the goal of entering operant
2. Accomplish the goal of typing operator
3. Accomplish the goal of entering operant
4. Report Goal Accomplished

B. Selection rule for the goal of entering operant
1. If the operant is simple, then type the variable
2. If the operant is complex, then accomplish the goal of entering formula

C. Method for accomplish the goal of typing operator
1. If operant before operator is complex, press "()" key   until desired chunk highlighted

2. Type operator
3. If operant after operator is complex, press "()" key
4. Report goal accomplished

Figure 3.11 A Cognitive Model for Using the Semantic Editor

The only perceptual difficulty for the semantic editor occurs in step C1 where users need to see whether the desired chunk is highlighted. As explained before, this checking is seldom needed; most of the time, the first '()' will highlight the chunk

desired. Also, this difficulty arises not as a deficiency of the editor but a deliberate design choice as explained. We have decided not to display the structured formula as it is being keyed in, which will make the highlighting feedback easier to evaluate, and not to enforce the binary nature of the formula that will further eliminate the need to highlight larger chunks.
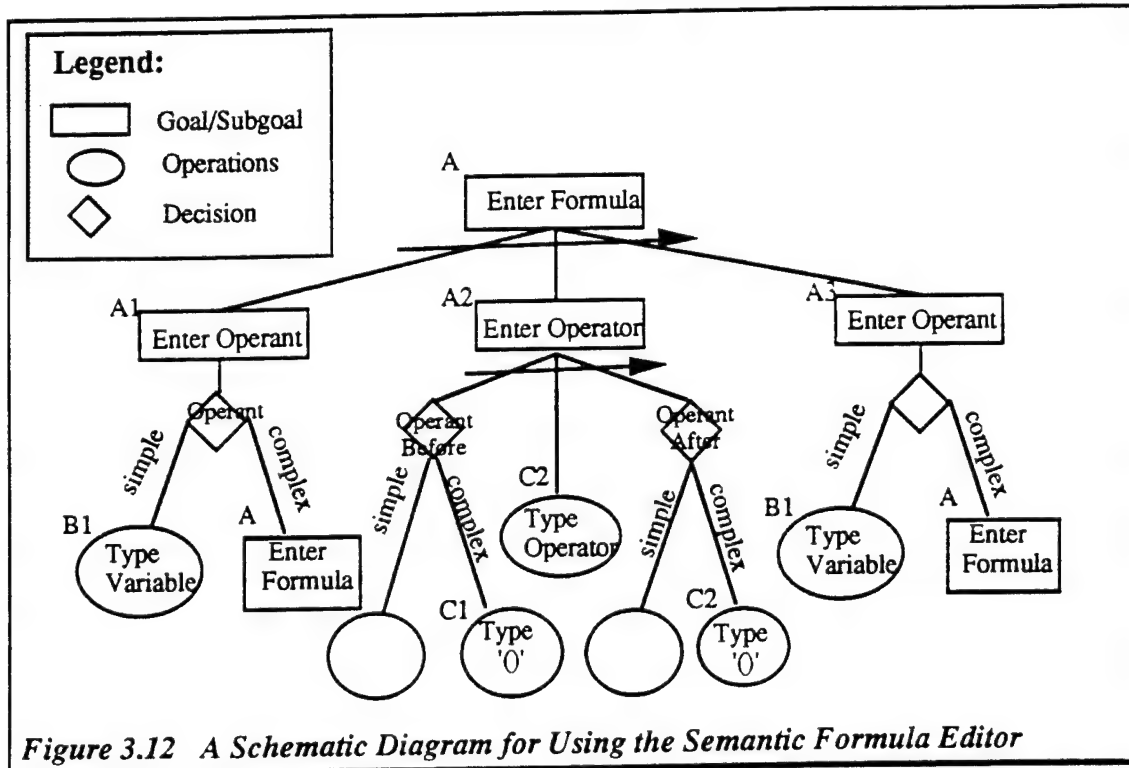
Figure 3.12 A Schematic Diagram for Using the Semantic Formula Editor

In the ideal situation of optimal performance like those modeled by GOMS, the above semantic editor will produce subgoals of three keystrokes each. The subgoals are the <O><A><O> chunks and this will give a keystroke time of **Mkkk** as predicted by the GOMS keystroke level model. Since there is no need for planning and tracking of these subgoals, the time predicted by a keystroke-level model will closely match the actual performance time. Whereas when using the linear editor, the prediction of a keystroke-level model will be way off as there is too much time for planning and validating. Also, due to cognitive overload, the task of keying complex formulas linearly may never be practiced enough to be skilled.

Although the production system predicts that the semantic editor will produce fewer errors and shorter performance time, we do not know users' performance when using the semantic editor. Users might have conceptual problems learning how to use the editor. Users might find the editor impossible to learn due to its novelty or users might find it difficult to see the recursive semantic structure in the formulas to be keyed in. We will now perform an experiment to test our model's prediction and the usability of the semantic editor, and the subgoal theory.

3.9   A Study to Compare the Cognitive Process when Using the Two Editors

The Objective

The objective of the experiment is to illustrate the subgoal theory by a direct comparison of the performance of subjects using the linear editor and the semantic editor to key in formulas. The performance in terms of time and error rate will be compared to the predictions of the cognitive models for using the two editors. The outcome will demonstrate how a simple redesign of the interface allows users to map the task directly into meaningful subgoals thus reducing extraneous subgoals and eliminating many cognitive bottlenecks.

The Subjects

Sixteen first year computer science students with limited computer experience were recruited from the National University of Singapore at the start of their first semester. They were paid an equivalent of US$10 each for participating in the experiment. Subjects were randomly assigned to one of the four conditions of the experiment. The conditions are the order of using the editors in the experiment and the

formula versions to be used with the editors to counter balance any possible practice effect.

The Physical Setup

The experiment was conducted in a computer lab with 16 IBM ATs with color monitors and extended memory. The computers are on four rows of tables with four computers in each row.

The Data Logging Programs

A keystroke capture program, 'RTCapture,' developed at the University of Michigan was used to time stamp each keystroke. The key capture program has a resolution of two milliseconds. The editors were programmed in Turbo Pascal and were able to capture the completed formulas in a text file for further analyses of the types and positions of errors. The screen was programmed to display only the formula the user was entering.

The Tasks

Each subject had to key in 20 formulas of varying complexity, similar to those used in the pilot study, using both the linear editor and the semantic editor. The subjects saw a different version of the 20 formulas when they switched editors. The two versions were developed to have identical structure but different cosmetic appearance by varying the variables and operators. The first four formulas were used as warm-ups and are not analyzed. Appendix D lists the formulas used in the experiment.

The Dependent Measures

The four dependent measures that will be investigated are (a) the time per keystroke to key in a formula, (b) the number of errors in a formula, (c) the number of extra keystrokes per formula, and (d) the position of parenthesis errors.

(a) The time per keystroke is the total time to key in a formula divided by the ideal number of keystroke to key in the same formula, i.e., when there is no backtracking to correct mistakes. The time per keystroke is a combination of time for cognitive activities (M) and key press (k). This is a gross measure of performance. We expect that the keystroke times when using the editor to key in formulas should not exceed two seconds (M + k). If the keystroke time is greater than two seconds, elements of planning and checking of subgoals, or extra keystrokes must be involved.

(b) The number of errors in a formula is the tally of the missing or extra parentheses, and the missing or wrong variables or operators.

(c) The number of extra keystrokes is the actual number of keystrokes used minus the ideal number of keystrokes for the particular formula entered. Extra keystrokes occur when users need to backtrack to correct mistakes or to insert a forgotten parenthesis.

(d) The position of parenthesis error is the level of embedding in the formula within the parenthesis pair involved. A parenthesis error is one where the parenthesis is either missing or extra. In the following examples, the square bracket denotes a parenthesis error. The errors can occur either at the right or left parenthesis positions. Only the left parenthesis errors are shown in the examples. For example: a position one error, [1+2); a position three error, [((1+2)*3)/2)^2; etc.

The time per keystroke, the number of errors, and the number of extra keystrokes are a set of items which trade off. A subject who was careful in planning and checking would have fewer errors but the time per keystroke would be long. Similarly, subjects that had fewer errors could be using lots of extra keys to backtrack to correct mistakes.

The Independent Variables

The experimental design is 2x2x2x4 within subject design with **editor**, **skewness**, **decay**, and **level of embedding** as the independent variables. Each is described and illustrated with examples below. There were also two between subject variables, the order of editor usage and the formula version, but these were counterbalanced and not included in the analyses as we did not hypothesize any effects for them.

Editor: the linear editor versus the semantic editor as described.

Skewness of Formula: left skewed versus right skewed parentheses cluster. An example of left skewed formula is $(((1+2)*3)^4)/5$; an example of right skewed formula is $4/(4^(1*(2+3)))$. We want to see whether it is the planning of opening parentheses or the validating of closing parentheses that causes problems. Our hypothesis is that planning and balancing parentheses are equally difficult. Both arise due to the mismatched of object structures in the task space and system space leading to extra subgoals.

Decay before Closing parenthesis: long decay versus short decay. Decay has to do with how long a subgoal needs to be kept in working memory. In particular, in using the linear editor, the subgoal of closing parenthesis will be kept in working memory until all intervening variables and operators have been keyed in. An example of short decay before closing parenthesis is $(1*(2+3))/4$; an example of a long decay is $(1*(2+3)+5^6)/4$. The literature is unclear whether memory overload is caused by displacement or decay of items in the memory. The results will help us identify whether long delay will cause loss of information from memory.

Level of embedded parentheses: two, three, four or five levels of embedding. An example of a formula with two levels of embedding is $(1*(2+3))/4$; an example of one

with five levels is $((((1+(2*3))*6)^3)/6)^2$. This is a manipulation of the complexity of the formulas thus the memory load of subgoals to balance closing parentheses.

Appendix E gives a listing of the formulas used and their classification.

The Experimental Procedure

The experimenter introduced the experimental procedure and demonstrated the two editors using overhead projection of a PC screen in a 40 minute session. Subjects then had a practice session for the Linear Editor and a practice session for the Semantic Editor where they were led through a series of examples with increasing complexity. At the end, they had to key in four practice formulas correctly twice to reach criterion. Appendix F lists the practice session instructions and the practice formulas.

Subjects were then randomly divided into four groups with each group taking up one of the four rows of tables. Each subject had one computer to himself or herself. Half the subjects then started the actual task with one editor while the other half started with the other editor. There were two versions of the formula booklet as described in the task description: Version A and Version B. To counter-balance the possible order-effect of using different editors and formula versions, half the subjects using a particular editor were presented with version A, the other half, version B. After they had entered all the formulas using one editor, they proceeded to enter another 20 formulas from a different version of the formula booklet using the other editor. However, before they entered the 20 formulas using a different editor, they had to key in the four practice formulas correctly twice again using the switched editor so that there were minimal residual effects from using the previous editor. In the actual trials, subjects entered 20 formulas presented one on each page in a booklet. The first four formulas will not be analyzed as they serve to bring the subjects up to speed in addition to the four practice formulas.

3.10   The Hypotheses

We expect that subjects will perform better when using the semantic editor compared to when using the linear editor. In particular, we expect the independent variables to affect the linear editor but not the semantic editor (see Figure 3.13). We will discuss the hypotheses (stated only in their alternate forms) under two sections: (a) the positions of errors, and (b) the effects of the levels of embedding, skewness, and decay on the dependent measures. We discuss the last three variables together since they are tradeoffs.

Figure 3.13   The Hypothesized Interaction Effects of the Independent Variables on the Dependent Measures
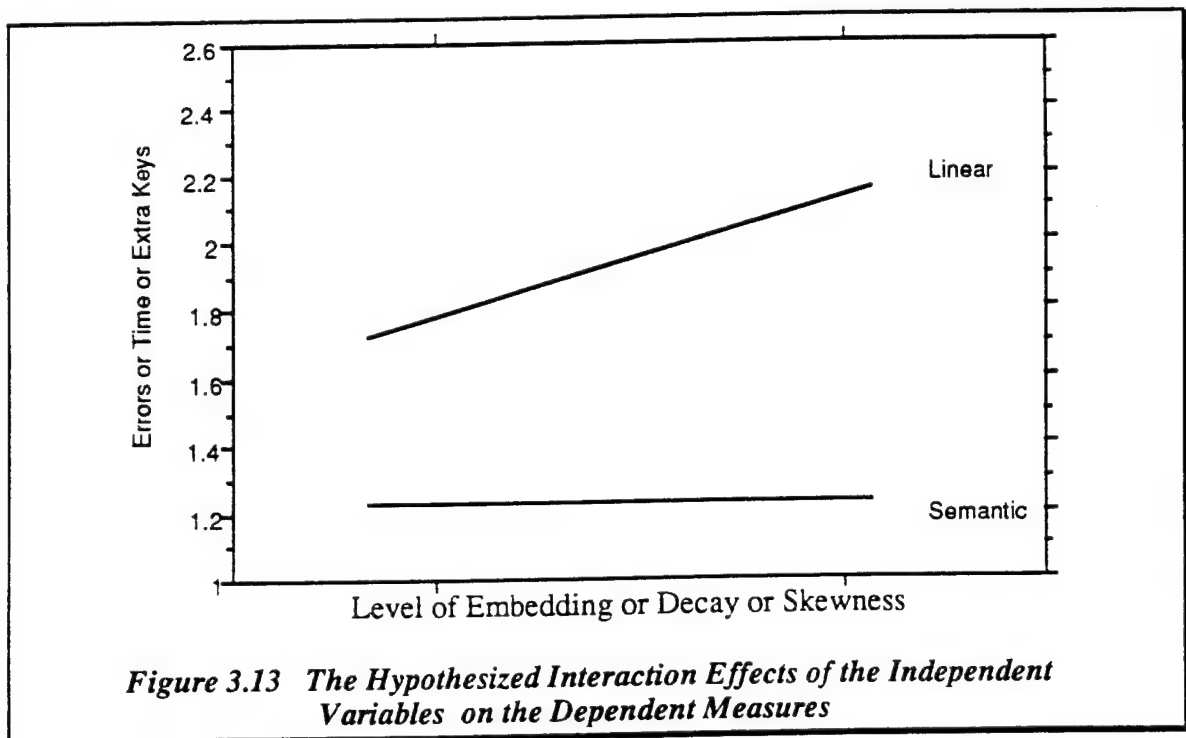
The Positions of Parenthesis Errors

For the linear editor, we expect that there will be more errors at parenthesis positions where the level of embedding within the parentheses increases[1]. As the

_____

[1]See page 91 discussion on the *Position of Parenthesis Errors* dependent measure for examples.

embedding level increases, there will be more subgoals for planning and checking to parse the embedded hierarchical formula to a linear structure to see how many opening or closing parentheses are needed. Parsing requires working memory to keep track of intermediate chunks. Also, as the level of embedding increases, the memory load will increase to keep track of the increasing number of subgoals for closing parentheses. When the memory load is exceeded, there will be errors. This will not be the case for the semantic editor as there are no extra subgoals to translate the mismatched object structures and therefore no planning, tracking, and checking of these extra subgoals.

Hypothesis 1A: For the linear editor, the number of parenthesis errors will increase at parenthesis positions with higher levels of embedded formulas.
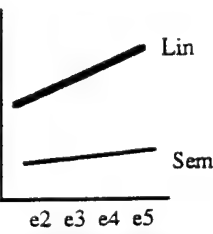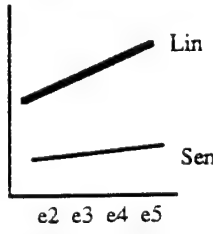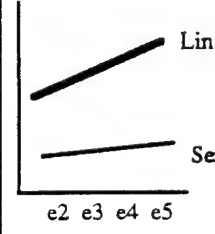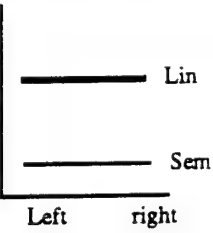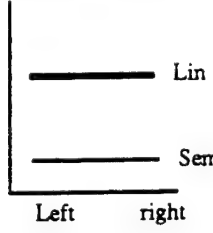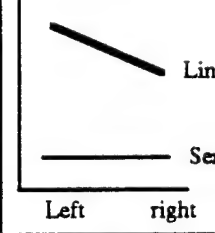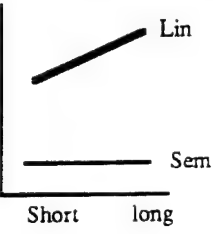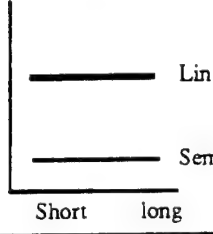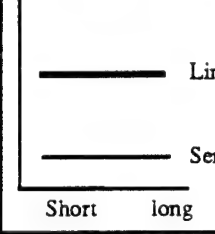
Hypothesis 1B: For the semantic error, the position of parenthesis error will not be correlated with the level of embedded formula.

The Effects of Level of Embedding, Skewness and Decay

The hypotheses for the effects of the four independent variables on the three dependent variables are summarized in Table 3.1. The columns present the dependent measures and the rows present the independent variables. For the first independent variable, Editor, we expect the semantic editor to perform better for all three dependent measures.

For the second independent variables, the Level of Embedding, which manipulates memory load, we again expect the semantic editor to perform better. However, the interaction effect is what we are looking for. We expect increasing levels of embedding will deteriorate the linear editor performance increasingly but not so for the semantic editor. However, the three dependent measures are tradeoffs and subjects can reduce errors by taking more time to plan and check or to back track to correct mistakes.

Consequently, we expect level of embedding to affect at least one out of the three dependent measures.

	The Dependent Measures		
The Independent Factors	No of Errors	Time Per Key (seconds)	Extra Keys
(H2) Editor	Semantic Better	Semantic Better	Semantic Better
(H3) Level of Embedding Level x Editor	*[graph: Lin rising, Sem rising slightly; x-axis e2 e3 e4 e5]*	*[graph: Lin rising, Sem rising slightly; x-axis e2 e3 e4 e5]*	*[graph: Lin rising, Sem rising slightly; x-axis e2 e3 e4 e5]*
(H4) Skewness Skewness x Editor	*[graph: Lin flat, Sem flat; x-axis Left right]*	*[graph: Lin flat, Sem flat; x-axis Left right]*	*[graph: Lin falling, Sem flat; x-axis Left right]*
(H5) Decay Decay x Editor	*[graph: Lin rising, Sem flat; x-axis Short long]*	*[graph: Lin flat, Sem flat; x-axis Short long]*	*[graph: Lin flat, Sem flat; x-axis Short long]*

Table 3.1   The Hypotheses for the Effects of the Independent Variables on the Dependent Measures

For Skewness, we do not expect any main or interaction effect for the first two dependent measures since we have stated that planning for open parentheses is as difficult as balancing parentheses. But, we do expect an interaction effect for the independent measure of the number of extra keys. Left skewed formulas will make planning for opening parentheses difficult. Subjects will forget to open parentheses and later when they realize the mistake and backtrack to insert the opening parentheses.

For Decay, we expect that a long embedded formula will likely cause decay of the subgoal to balance parentheses. This will lead to more errors for formulas with longer decay when using the linear editor but not when using the semantic editor. Decay is unlikely to affect the other two dependent measures. However, subjects could reduce the number of errors by increasing the checking time if they realize that the formula does not look right.

3.11   The Results of the Formula Editor Experiment

In general, the results confirm our hypotheses that the semantic editor can reduce the cognitive difficulties when used to enter formulas. The semantic editor, compared to the linear editor, reduces the number of errors committed, the keystroke times and the number of extra keystrokes. Also, the performance when using the semantic editor, unlike that when using the linear editor, is not affected by the levels, the skewness, and the length of the embedded formulas.

Despite its novelty, the semantic editor was easy to learn. Subjects on the average only took 20 minutes longer to go through the practice formulas when learning the semantic editor as compared to that when using the linear editor. The extra time is reasonable since subjects have seen the linear editor before but not the semantic editor. Also, subjects commented after the experiment that the semantic editor was easier to use compared to the linear editor for entering formulas.

Due to computer crashes, only fourteen out of the sixteen subjects' data are captured. This will not pose a problem for balancing the treatment conditions since every independent variable is a within-subject variable. The between-subject factors, the editor order and the formula version, do no affect the results significantly. Since they are counterbalanced, they are collapsed into the within-subject factors and are not analyzed. We will discuss the results according to the format of the section on hypotheses.

3.11.1 The Positions of Parenthesis Errors

The Semantic Editor

Out of the 224 formulas that the 14 subjects keyed in using the semantic editor, there were only a total of nine errors. Out of the nine errors, six are accounted for by three missing parenthesis pairs and three missing variables (0.5) for creating square root operations. For example, subjects did not transcribe $\sqrt{1+2}$ to $(1+2)^{\wedge}.5$ but left it as $1+2$. This can be attributed to slips since the visual cue for the need to create a square root operation is not obvious. However, since there are a total of 112 square root operations in the formulas that the subjects keyed in, three missing operations out of a possible 112 is negligible. The remaining three missing pairs of parentheses were simply left out by the subjects. It could be that subjects forgot to create the complex operants due to unfamiliarity with how the semantic editor works, or due to the difficulties in seeing the complex operant structure in the linearly displayed formulas on the screen.

The positions of the nine errors are also not correlated to the level of embedding thus ruling out memory load or visual load as the cause. Table 3.2 gives the location and the number of errors.

Position of error (Level of Embedding)	1	2	3	4	5
Number of Errors	1	3	2	2	0
Possible Errors	896	616	392	224	112

Table 3.2   The Positions and Number of Errors for the Semantic Editor

The Linear Editor

There were many more errors when subjects used the linear editor to enter formulas. There are a total of 93 errors in the 224 formulas that the 14 subjects keyed in.

There are 38 missing parentheses, 32 extra parentheses and seven missing variables. There are 16 formulas that are syntactically correct, i.e., the parentheses are balanced; but semantically incorrect, i.e., the balanced but misplaced parentheses created structurally different formulas. The almost equal number of missing parentheses and extra parentheses shows that planning and checking are equally difficult.

Figure 3.14   Numbers of Errors at Parenthesis' Locations
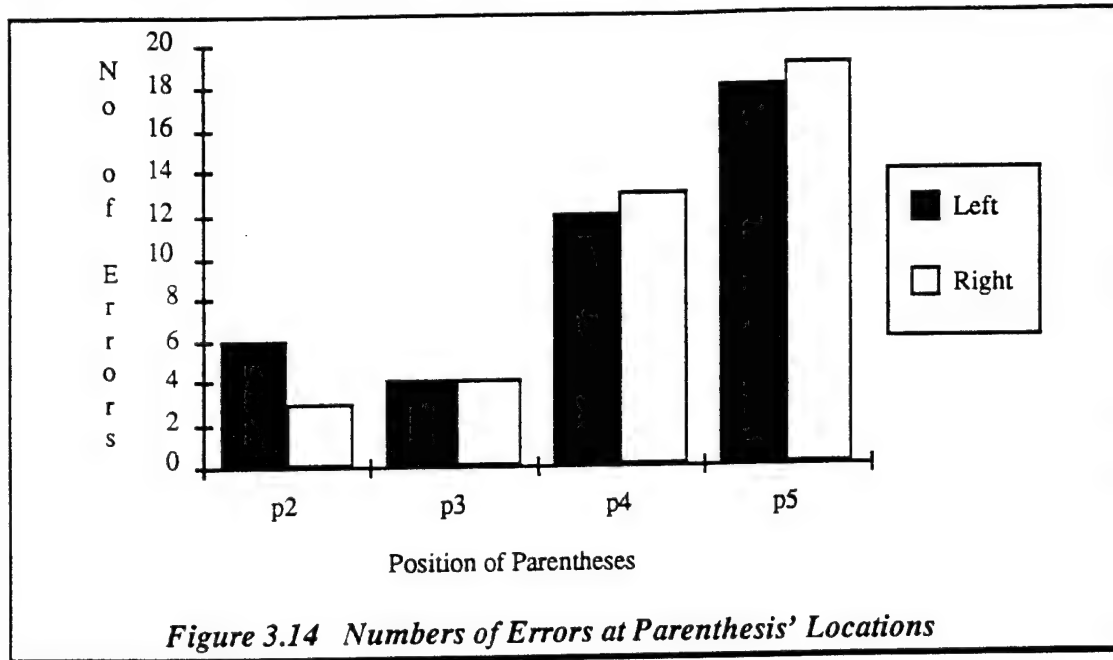
Figure 3.14 depicts the number of errors committed at the left and right parenthesis' locations according to how many levels of embedded formulas there are within the erroneous parentheses. The results show that subjects had more problems when there were higher levels of embedding [$F(3,39)=5.889$, $p=.0025$]. There were no errors at parenthesis' location when there was only one level of embedded formula, e.g., $(1+2)$. The cognitive model in Figure 3.7 predicted this phenomenon: higher embedding level leads to higher memory load that results in loss of information (the subgoal to balance parentheses) from the working memory. Also, we see from Figure 3.14 that there are about equal number of errors in both the left and right parentheses locations [$F(1,39)=.548$, $p>.05$]. This strongly suggests that balancing parentheses is not the only cognitive difficulty. This implies that a syntactic editor, like those found in some LISP

editors, that will balance any unbalanced parentheses will not solve all the cognitive problems when entering formulas linearly. Subjects had difficulties in planning for the subgoals to open parentheses by parsing the hierarchical formula structure to a linear string structure. The cognitive model again predicted this difficulty.

3.11.2 The Effects of Level of Embedding, Skewness and Decay

Table 3.3 depicts the results of the ANOVA on the effects of the four independent variables on the three dependent measures. The results, the directions and the significances of differences, all turned out as predicted by the hypotheses. We will discuss below the results organized by the independent variables.

Editor

The Semantic editor was superior in all three dependent measures. The time per keystroke for the linear editor of 2.317 is greater than that predicted by a GOMS keystroke level model of 2 seconds (M+k). There must be many extra cognitive activities other than mental preparation (M) when using the linear editor. Thus, this confirms our hypothesis that the keystroke-level model cannot account for the long time when using the linear editor to enter formulas. The planning, and evaluation of the extra subgoals to open and close parentheses must be accounted for.

The drastic difference in extra keystrokes for the two editors (.982 versus 16.527) can be attributed to subjects having difficulty planning for the subgoal of opening parentheses when using the linear editor. They often resort to using cursor keys to backtrack to insert forgotten opening parentheses.

There is more backtracking and many more errors when using the linear editor. Since all three dependent measures shown significant effects, the difficulty in using the linear editor was not overcome by tradeoffs among these three dependent measures. That

is, subjects still made more mistakes when using the linear editor despite spending more time in planning and checking and backtracking to correct mistakes.

The Independent Factors	The Dependent Measures		
	No of Errors	Time Per Key (seconds)	Extra Keys
Editor	Semantic = .045 Linear =.321 $F(1,13)=12.333$ p=.0038*	Semantic =1.289 Linear =2.317 $F(1,13)=228.535$ p=.0001*	Semantic = .982 Linear =16.527 $F(1,13)=28.320$ p=.0001*
Level of Embedding	e2=.125 e3=.107 e4=.250 e5=.250 $F(3,39)=3.254$ p=.0318*	e2=1.747 e3=1.556 e4=1.887 e5=2.021 $F(3,39)=19.733$ p=.0001*	e2=6.643 e3=8.098 e4=10.107 e5=10.170 $F(3,39)=2.153$ p=.1092
Level x Editor	$F(3,39)=2.940$ p=.0450*	$F(3,39)=6.622$ p=.0010*	$F(3,39)=2.018$ p=.1273
Skewness	left=.170 right=.196 $F(1,13)=.138$ p=.7158	left=1.811 right=1.795 $F(1,13)=.055$ p=.8180	left=10.63 right=8.98 $F(1,13)=6.148$ p=.0276*
Skewness x Editor	$F(1,13)=1.322$ p=.2692	$F(1,13)=.275$ p=.6090	$F(1,13)=7.299$ p=.0181*
Decay	short=.161 long=.205 $F(1,13)=.853$ p=.3725	short=1.769 long=1.873 $F(1,13)=.991$ p=.787	short=8.272 long=9.237 $F(1,13)=.991$ p=.3377
Decay x Editor	$F(1,13)=.582$ p=.4591	$F(1,13)=.868$ p=.0386*	$F(1,13)=.868$ p=.3685

Table 3.3 The Summary of ANOVA of the Effects of the Independent Variables

The Level of Embedding

The results in Table 3.3 show that the main effects for the level of embedding are significant for the number of errors and time per key. However, we need to test the simple effects to see whether the level of embedding only affects the linear editor and not the semantic editor as stated in Hypothesis H3 in Table 3.1. A pre-planned contrast analysis is used to test the effects of level of embedding on the linear editor and the semantic editor separately. A contrast with coefficient of -2, -1, 1, 2, is used for the the four level of embedding, i.e., we expect that higher level of embedding will lead to poorer performance.

The contrast analyses reveals results that confirm Hypothesis H3 in Table 3.1 entirely. The three dependent measures increase with the level of embedding for the linear editor, but not for the semantic editor. For the number of errors, the simple effect of the level of embedding on the linear editor is significant [$F(1,39)=14.534$, $p=.0005$], while that on the semantic editor is not [$F(1,39)=0.0$, $p=1.00$]. For the time per key, the simple effect of the level of embedding on the linear editor is significant [$F(1,39)=36.49$, $p=.0001$], while that on the semantic editor is not [$F(1,39)=.556$, $p=.4563$]. For the number of extra keys, the simple effect of the level of embedding on the linear editor is significant [$F(1,39)=10.72$, $p=.0022$], while that on the semantic editor is not [$F(1,39)=.001$, $p=.9509$].

Since the level of embedding is a measure of the memory load to balance closing parentheses and to plan for opening parentheses, the significant effects of the level of embedding on the three dependent variables suggest that the memory load is a real problem for the linear editor users.

Skewness

The results in Table 3.3 and from simple effect tests confirm Hypothesis H4 in Table 3.1 on the effects of skewness on the dependent measures entirely. That is,

skewness only affected the number of extra keys when using the linear editor. Skewness has no effects on the number of errors and the time per key. Simple effect tests also reveal that skewness has no effect for these two independent variable for both editors [all four cases p>.3]. However, for the number of extra keys, the simple effect of skewness on the linear editor is significant [F(1,13)=13.899, p=.0033], while that on the semantic editor is not [F(1,13)=0.053, p=.8222].

This shows that balancing parentheses is not the problem, planning for opening parentheses is equally demanding. The mean number of extra keystrokes for semantic editor (left, right) are .742 and 1.223, whereas that for linear editor (left, right) are 20.304 and 12.75 respectively. When there is a cluster of left parentheses, subjects had difficulty planning the subgoals to open parentheses due to perceptual and memory overload.

Decay

Again, the results in Table 3.3 and from simple effect tests confirm Hypothesis H5 in Table 3.1 on the effects of decay on the dependent measures entirely. Although Table 3.3 shows that decay does not affect the number of errors when the two editors are considered together, a simple effect test shows the results as we hypothesized. Decay affects the number of errors for the linear editor [F(1,13)=2.070, p=.0442] but not the semantic editor [F(1,13)=.129, p=.3232]. No other significant simple effects is found for the other two dependent measures as hypothesized in Table 3.1. The results show that decay of working memory is a problem as well as working memory load.

3.12   Discussion of Results

The results suggest that subjects were forced to adopt many unnatural, nested subgoals when the object structures manipulated are different in the task space and the system space. This results in the task acquisition time and evaluation time overwhelming

the keystroke time, thus making the GOMS keystroke level prediction inaccurate. With an average keystroke time of 2.317 seconds, it seems that practically every keystroke constitutes a subgoal by itself.

Since there were no trade off between the number of errors and the time per key, subjects using the linear editor made many errors despite careful planning and checking. The slow time per key indicates that subjects generally cannot remember when to close the parentheses. They engaged in checking to try to ensure that the formula was correctly keyed in. This is reasonable as the memory load is easily exceeded with even a low level of embedding due to the use of working memory to translate the hierarchical structure to the linear structure.

The results also show that the key-stroke level model assumptions of the independence between task acquisition and task execution is invalid here. A KLM assumption states that making the command more efficient does not affect the acquisition time. We show otherwise here. A good interface that allows the formation of a subgoal structure that reflects the task structure will greatly reduce the task acquisition and task evaluation time.

Also, the results show that when using the linear editor, a user may never become skilled due to the excessive memory and perceptual load. Thus, we may never be able to apply the KLM in its current form since the users may never be skilled.

Finally, it illustrates why a simple qualitative difference in the interface design makes a big differences in performance although the number of keystrokes remain the same. It is the subgoal structure that explains the performance differences and not the number of keystroke as in the current KLM.

3.13 Conclusion

In this chapter, we illustrated the subgoal theory with a simple application of entering formulas using different editors. When using a linear editor to enter formulas, users employ many extra subgoals to bridge the task-system mismatch as they operate on different structures. These extra subgoals created many cognitive difficulties of planning, tracking, and validating these extra subgoals that do not reflect the task structure. The subgoals are short, unnatural, and often only contain one keystroke.

We then designed a semantic editor that operates on the <O><A><O> structures of formulas that alleviates many cognitive difficulties when using the linear editor. The subgoal structure when using the semantic editor is natural, reflects the task structure, and does not need much planning, tracking, and validating.

This empirical study illustrates the importance of designing an interface that allows the formation of good subgoal structure. The subgoal structure must reflect the task structure, as well as the way the user thinks of the task; it must be within the working memory constraint when decomposed and easy to acquire and validate. The semantic editor shows that we do not always need a fancy interface to make an interface easier to use, just some careful thoughts about how users think of the task structure.

The next chapter describes an experiment to show again that extraneous subgoals cause poor performance. However, this time the extraneous subgoals are generated to overcome the mismatch when the interface does not allow the user to execute the subgoals in the order conceived.

CHAPTER 4

THE LOTUS MENU EXPERIMENT

This chapter describes a Lotus menu traversal experiment in which subjects execute spreadsheet modification tasks using the Lotus hierarchical menu system. The objective of the experiment is to further illustrate the theory in Chapter 2 that extra subgoals are used to bridge task-system mismatch leading to degraded performance. The task-system mismatch here is operationalized as an interface that does not allow users to execute actions in the order they are conceived in the task space. It shows how the interface can affect the way users think about the task, and subsequently how the task is decomposed into subgoals, and thus affecting the execution time and error rate. This violates the GOMS keystroke-level model's assumption that the execution time is independent of how the task is acquired. It also demonstrates that consistency within interface alone is important but consistency of interface with how users think of the task also can affect performance.

In this experiment, we manipulate how subjects decompose a task goal into subgoals by manipulating the grammatical structure of the menu and instructions presented. We present two versions of the menu, the original Lotus menu with an inconsistent structure and a revised menu with a consistent <object><action> structure. We also present two versions of instructions, one version has an <object><action> structure that matches the revised Lotus menu and the other version has an <action><object> structure. We expect that subjects given the consistent revised menu can form a consistent and efficient subgoal structure to explore the menu and execute the

commands. When the instruction structure matches the menu structure, users can decompose the task goal into subgoals that match the order of hierarchical items on the menu. When there is a mismatch, users will need an extra subgoal to buffer the out-of-order subgoals. This extra cognitive operation will degrade performance.

The following sections in this chapter present the experiment in detail. We first describe the spreadsheet modification tasks, then we describe the lotus menu structure, the rearrangement of the menu items, and the instructions for the tasks. We then present a cognitive model of the menu traversal task. Finally, we describe the experimental setup, methodology, and the hypotheses to be tested.

4.1 The Spreadsheet Modification Tasks

There is a set of common modification tasks when creating or modifying spreadsheets. This set of tasks is domain independent; they primarily change the appearance of the spreadsheet regardless of the content of the spreadsheet. Examples of such tasks are: setting the column width, changing the number format, copying and moving a block of cells, changing the alignment of the cell contents, etc.

We chose Lotus 123 as the spreadsheet program to execute these tasks as it has the largest installed base of users. However, the results can be generalized to other spreadsheet programs or any programs that use a hierarchical menu to issue commands. Since we are only modeling how people compose commands given a task description and a hierarchical menu structure, any program that uses a hierarchical menu system will suit our purpose. The use of the Lotus spreadsheet program is not critical to our findings.

On the other hand, there are some advantages of using a spreadsheet program in the experiment. First, spreadsheet programs are not used as much as word processing programs in HCI research. This will allow findings here to generalize to more than word processing programs alone. Second, since spreadsheets are not as widely used as word

processors, it will be easier to find naive subjects. We need naive subjects in this experiment so that their domain and application knowledge will not confound our findings.

The following ten spreadsheet modification tasks are used in the experiment. They are the most common and frequent tasks in spreadsheet applications besides entering domain-specific information in the cells.

1. Adjust all the columns' width at once.
2. Adjust the width of a few columns.
3. Set all the contents of the spreadsheet to a desired format.
4, Set the contents of a block of cells to a desired format.
5. Erase the contents of a block of cells.
6. Copy or Move the contents of a block of cells.
7. Insert row(s) or column(s).
8. Delete row(s) or column(s).
9. Align the contents of the entire spreadsheet.
10. Align the contents of a block of cells.

The method to execute these tasks is to issue a command through the Lotus menu. We next discuss the Lotus menu structure and the mechanism for issuing commands though the menu, and introduce the two versions of the menu used in the experiment.

4.2 The Structure of the Lotus Menu

All tasks in the Lotus program are executed by commands issued through a hierarchical menu except entering cell contents and navigating. A hierarchical menu system is one in which complex commands are issued through picking items from a multi-level menu tree. With the proliferation of more complex software, hierarchical menus are now used widely so that many commands can be organized in logical clusters. Users need not memorize the commands; they can explore the functions that they do not

know the execution method exactly. To issue a command through a hierarchical menu, users choose one item from the first level menu and then choose one option under this item and proceed to the last level of this branch of the menu tree.

To invoke the Lotus menu, users press the '/' key. The top screen in Figure 4.1 shows what the first level Lotus menu looks like. The first line of the menu (second line on the screen) gives the options at the current level. The second line of the menu gives the options that are available if the current highlighted item on the first line is selected. This is a 'look ahead' feature and is essential for menu exploration. If there are no options available for the current highlighted item, i.e., the item is at the last level of the menu tree, the second line of the menu gives a brief explanation of the function of this highlighted item instead.

When the menu is invoked, the first item on the first level menu is highlighted. To highlight a different item, users employ the left or right cursor key to move to the desired item. As the cursor key is pressed, the next item on the same menu level in the direction of the cursor will be highlighted. An item is selected by pressing the 'return' key when it is highlighted. Instead of using cursor key to highlight and then hitting the 'return' key to select an item, the user also can type the first letter of the item to select it. Once the item is selected, the options under this item will move up to the first line of the displayed menu and the users continue to proceed down the branch of the menu tree. The users can backtrack up a level by pressing the <escape> key if they venture down the wrong branch.

A complete command sequence in Lotus requires anywhere from one to four levels of menu traversal followed by typing in the parameters requested if necessary. Examples of parameters can be the desired width of the column, the number of decimal points for number formats, etc. For example, to format cells, users first select *Range* from the first level menu, *Format* from the second level menu, and the type of format

desired from the third level menu (e.g., *Currency*). The users will then be prompted for

the desired number of digits after the decimal point. The whole command sequence is

completed after users supply the parameters requested and hit the final <return> key.

Figure 4.1 The Menu Structure and Menu Traversal in Lotus
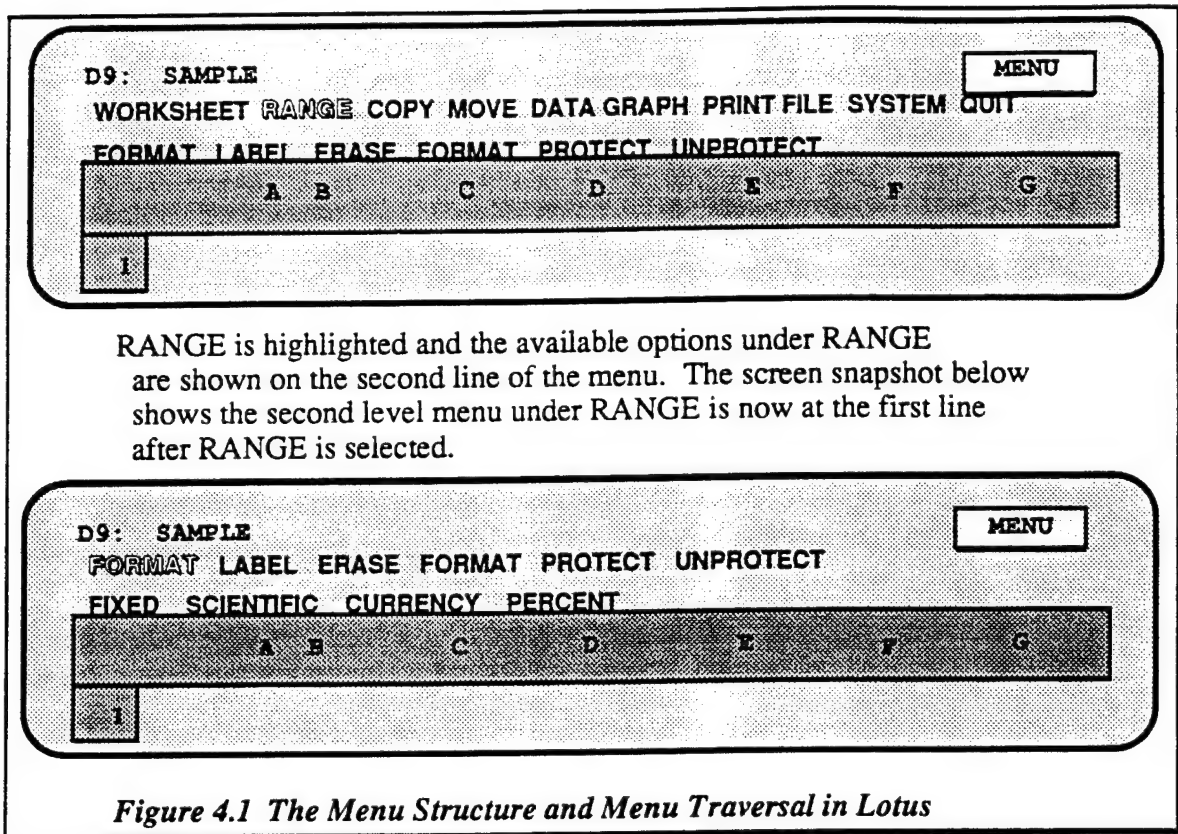
As explained earlier, the users can type the first letters of the menu items to issue

the command instead of using the cursor key to highlight the item then press the <return>

key to select it. For the example above, users can issue the sequence of keystrokes

'/RFC' to format the cell contents to "currency". This provides a short cut for expert

users engaged in rule-based behavior. They just need to retrieve the sequence of

keystrokes needed to issue the command without conscious awareness of what command

the individual keystroke corresponds to. However, we expect our novice subjects to

engage in knowledge-based behavior initially by actively engaging in problem solving

and that they need to know whether each menu item will bring them closer to the desired

task goals. Thus, our subjects have to tackle the menu items on different levels one at a time. They will either use the cursor method or will need to tackle each key-binding for the menu item as a separate subgoal.

The Menus Used in the Experiment

In the experiment, we only present a subset of the entire Lotus menu tree to the subjects. We only present items on the menu that are related to spreadsheet modification functions and file functions. The graphics, printing and database functions are pruned from the menu presented. This way, the naive subjects can get up to speed quickly without being bogged down by an unduly complicated menu structure.

The Original Menu

Figure 4.2 shows the first version of the menu used in the experiment. This version retains the structure of the Lotus menu (version 2.2) with items that are not relevant to our 10 modification tasks trimmed out. The database, graphics, printing and systems functions have been taken out.

The Lotus menu organization is hybrid. It is organized both by functions and by frequency of use. On the first level menu, the first item 'Worksheet' has to do with functions that affect the appearance of the spreadsheet, e.g., column width, alignment, cell format. The second item 'Block'[1] deals with operations having to do with a block of cells, e.g., format, alignment, etc. Copy and Move ought to be under the Block options but are moved up to the first level as they are the most frequently used commands in any

---

[1] 'Range' in the original menu. The letter R will be used in the first level menu by the menu item Row in the new menu described later. We cannot use Range since two items will then start with the same letter R. This makes using the first letter of a menu item to select the item difficult.

spreadsheet program. The File option deals with file operations like saving and retrieving, and changing directory.

Figure 4.2 The Subset of the Original Lotus Menu
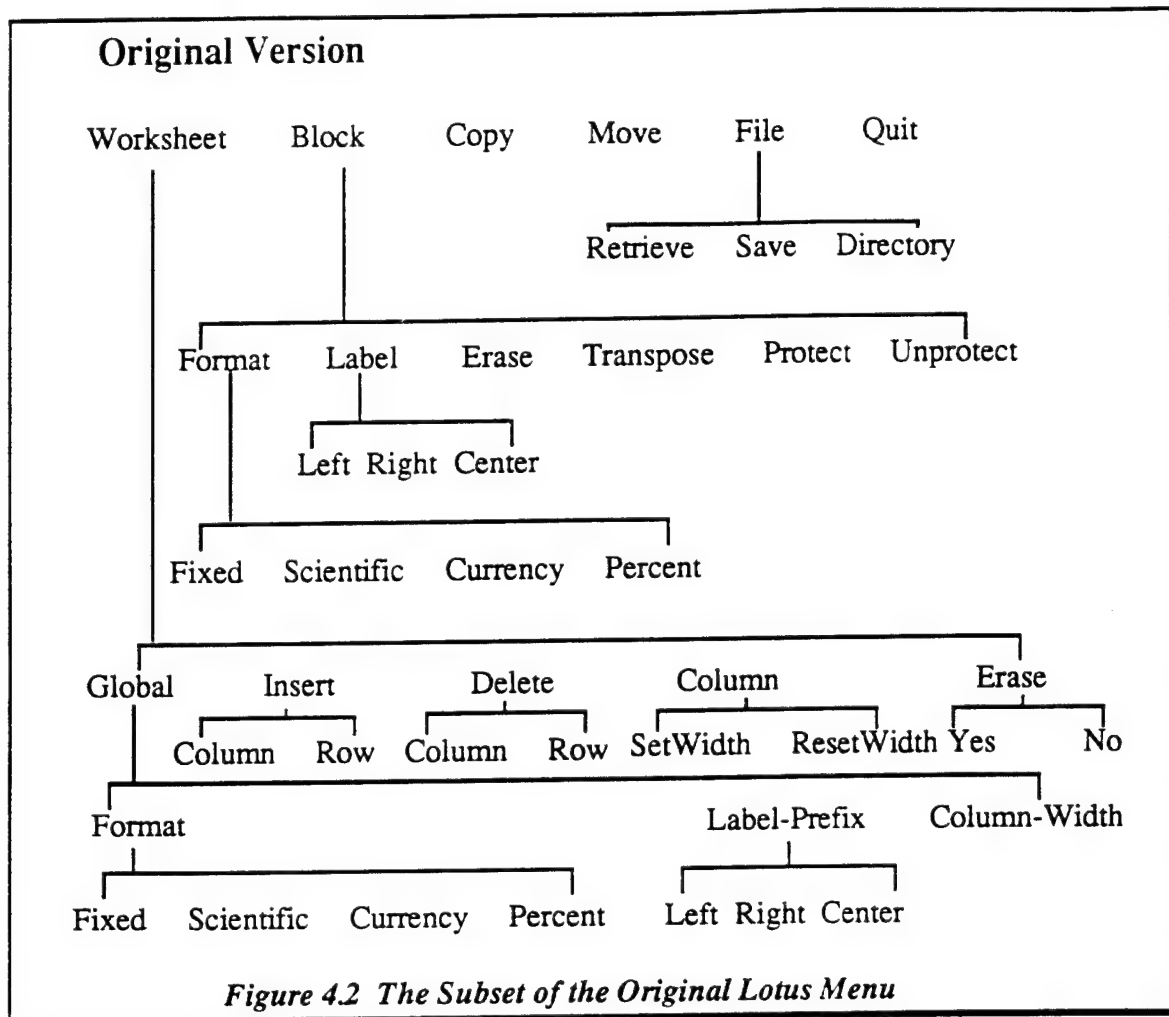
We can see that the structure of the menu tree is inconsistent in the order of components. There are actually three types of menu items: (a) object (O), (b) the action (A) to be performed on the object of interest, and (c) arguments needed to complete the command. If we exclude the argument specification at the leaf of the menu tree, users see one of four menu structures on the original Lotus menu: OOA, OAO, OA, A.

This organization is difficult to handle for novices since it is inconsistent in structure. The menu structure does not matter as much to the experts in rule-based behavior as they retrieve the equivalent command key sequence regardless of the

structural meaning of the key sequence. However, this inconsistent menu structure will cause problems for novices in knowledge-based behavior as they cannot explore the system space effectively by having a consistent subgoal structure. For example, they cannot have a consistent subgoal structure to look always for the object of interest in the first level menu, then to have a second subgoal to look for the action to perform in the second level menu, or vice versa. The current Lotus menu might save a keystroke or two for expert subjects who have memorized the key sequence to execute a command, but it will not be easy for novices to find the correct command let alone memorize it due to its structural inconsistency. The ill-structured Lotus menu could be the reason for subjects in our previous longitudinal study (in preparation) not being able to memorize all the key sequences and still committing many errors even after two years of extensive use.

The Object-Action Menu

There are many ways to restructure the Lotus menu to help novices as well as experts. We chose to restructure the menu such that the first level menu lists the objects of interest, the second level menu lists the possible actions that can be performed on these objects, and the third level menu (if necessary) lists the argument to complete the command. The objects at the first level menu are Block, Column, Row, Global and File. Global is chosen when the object of interest is the whole spreadsheet, e.g., changing all the columns' width. Verbs are used at the second level menu to denote the actions that can be done with the first level objects, e.g., copy, delete, set width, etc. Figure 4.3 shows the restructured Lotus menu.

We choose to list 'Objects', instead of 'Actions' under the first level menu as there are too many actions and they will not fit on the screen. Having so many items on the first level menu will make the menu too broad and thus make the search difficult. This will hamper subjects' knowledge-based behavior of exploring the menu tree. Also, we can think of the database function, the graphics function, and the printing function—

the three branches in the original menu not included in the revised menu — as objects (data, graphical, and printer objects) we want to manipulate. Thus, the resulting reorganization of the menu does not deviate much from the original version. They are almost equivalent in breadth, depth, and wording. However, we hope to demonstrate that this subtle menu items reorganization will lead to a significant difference in performance due to the different subgoal structure needed to traverse the two menus.

Figure 4.3 The Restructured Object-Action Lotus Menu
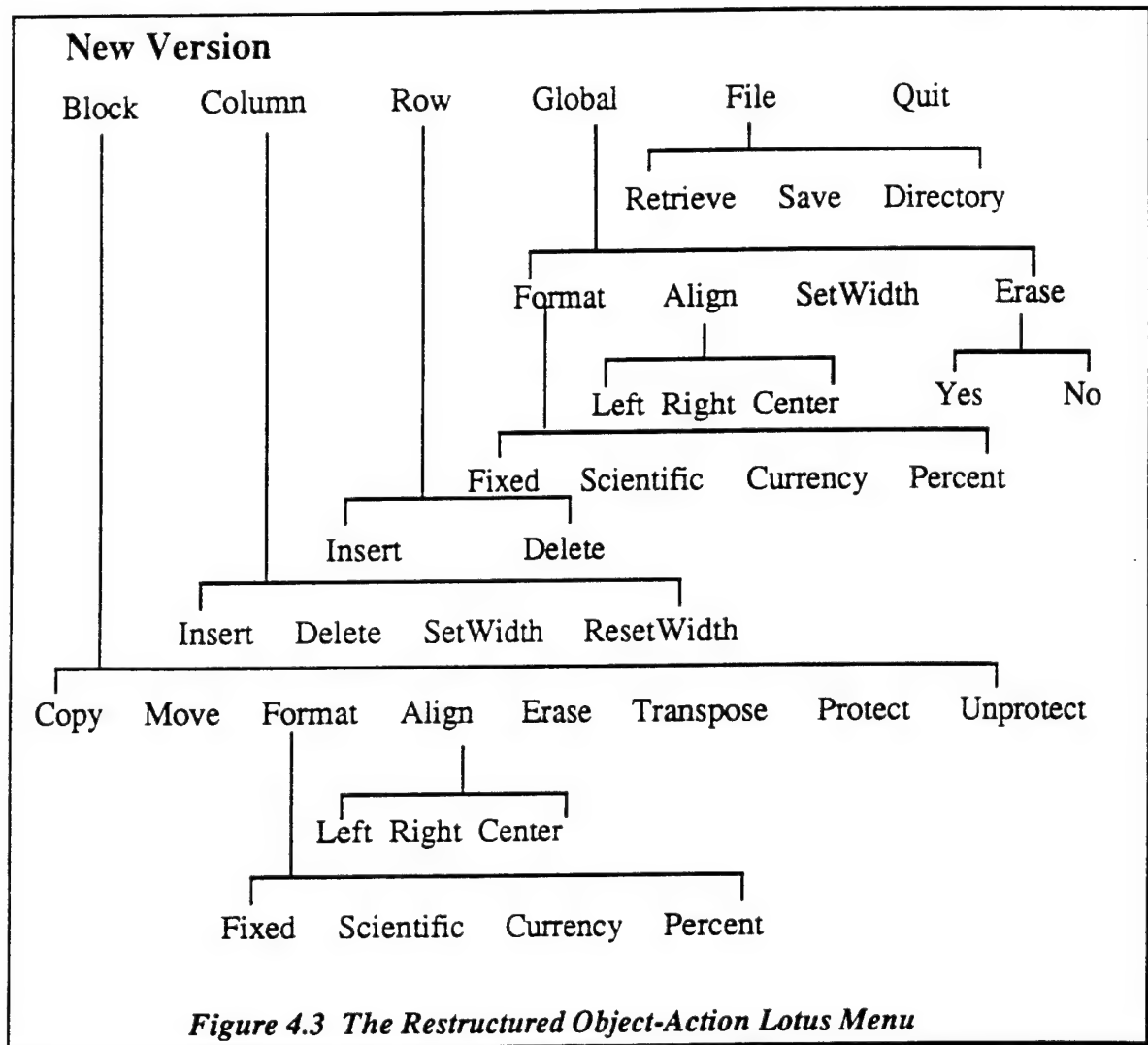
Both versions of the menu use the same wording for corresponding items except that we use *Align* in the new menu for *Label-Prefix*. Using the same wordings will ensure that the differences in performance are due to the menu structure change but not

due to the differences in the wording of the items. For *Align*, the original menu uses label-prefix which is not a verb thus violating the new menu structure of having a verb for action at the second level. We use 'Justify' in the instruction of the align cell content task so that the instruction does not bias against any menu type. Block instead of Range is used in both the new and old versions as the new version has Row in the first level menu making the use of Range impossible with two items starting with the same letter. This is not a problem as our subjects have not used spreadsheet programs before and will not have preconceived ideas of what block or range means.

We expect users given the new Lotus menu to perform better than those using the old menu as they can form a more consistent subgoal structure to explore the menu. This will lead to faster learning, better performance, and fewer errors. To further manipulate how users will decompose the task goal into subgoals, we will manipulate the grammar structure of the instructions. We expect that a match between the instruction structure and the menu structure will further improve performance.

4.3   The Instructions for the Tasks

We use different grammar structures in the instructions to further control how subjects decompose the task goal into subgoals. There are two types of instructions where the instruction grammar will either fit that of the new menu structure or will be the direct opposite. One type of instruction always has the object listed first followed by the action, i.e., having the same grammatical structure as the new menu. This instruction type helps users decompose the task goal into subgoals of (a) specifying the 'Object', and (b) specifying the 'Action', that directly map to the new <O><A> menu. An example of this type of instruction is: "For Column A, set the width to 6".

The other type of instruction always lists the action first in a verb-object grammar. This type of instruction does not create a subgoal structure that maps directly to either the

old or the new menu structure. An example of this type of instruction is: Set the width of column A to 6. For users receiving this <A><O> instruction with the new <O><A> menu, they will need to create an extra subgoal to store temporarily the out of order subgoal of specifying the action first. The users execute the subgoal of specifying the object, then retrieve the stored subgoal of specifying the action. This extra subgoal will lengthen performance time and increase opportunity for error. Since the original Lotus menu has an inconsistent structure, we need to analyze each task separately. In summary, tasks where the menu structure and the instruction structure fit will benefit subjects more in terms of learning, error rate, and performance time. We will explore this in detail in a later section on the hypotheses for the experiment.

Cognitive Model for Lotus Menu Traversal with <O><A> Menu and Instructions

  A. Method to accomplish the goal of issuing command
  1.   Accomplish the goal of reading the instruction
  2.   Accomplish the goal of issuing the command

  B. Method to accomplish the goal of reading instruction
  1.   Read the first component of instruction
  2.   Store first component of instruction in memory
  3.   Read the second component of instruction
  4.   Store second component of instruction in memory

  C. Method to accomplish the goal of issuing the command
  1.   Retrieve first component of instruction from memory
  2.   Find menu item corresponding to first component of instruction
  3.   Accomplish goal of selecting menu item
  4.   Retrieve second component of instruction from memory
  5.   Find menu item corresponding to second component of instruction
  6.   Accomplish goal of selecting menu item

Figure 4.4 Cognitive Model for Lotus Menu Traversal when Task-System match

---

We can use a cognitive model to model what we have said above about the cognitive processes involved in issuing a command through the hierarchical menu after reading an instruction. Figure 4.4 shows a partial GOMS model of the menu traversal

process for the new <O><A> menu after reading a matching <O><A> instruction. We will ignore the steps involved in specifying the argument for the command.

In the cognitive model above, we describe the instructions as having two components, one describing the object of interest and one describing the action to be performed. If the menu structure does not match the instruction structure, users will not find the menu item corresponding to the first component of the instruction in Step C2. Users will need an extra subgoal to store temporarily the first instruction component and later retrieve it after the second instruction component is executed. This extra cognitive effort will cause longer performance time and higher error rate. For the old menus, there will not be a simple cognitive model to describe the underlying cognitive process as the menu structure differs for each task type. This inconsistent menu structure will cause longer learning time and higher error rates. We can only model the use of the old menu with a simple cognitive model for rule-based behavior when the commands are well learned corresponding to a sequence of keystrokes for the initial letters of the command menu items.

The above model also can be modified when users have already memorized the content of the instruction. In that case, Step B above will be skipped entirely and users simply retrieve the correct component from long term memory (either the object or the action) in Step C1 and Step C4. The structure of the instruction will not matter much then as they are already stored in long-term memory. In this situation, the instruction manipulation may not achieve the results we want but we still expect the new menu to be better, in terms of performance time and error rates, than the old menu regardless of instruction type. Users can learn and form a consistent subgoal structure quickly when the menu structure is consistent.

In summary, we have operationalized the task-system mismatch in two ways. In the strong manipulation, we alter the Lotus menu structure. With the new menu

consistent <O><A> structure, users should be able to decompose the task goal into a consistent subgoal structure thus leading to better performance. In the weak manipulation, we try to influence how users think of the task by structuring the wordings in the instruction differently to match or mismatch the menu structure. We expect users given an inconsistent instruction-menu structure will need extra subgoals to bridge the mismatch thus leading to degraded performance. The second manipulation is weaker than the menu change as we expect users may be able to memorize the instructions after a few trials as they will be seeing the instructions repeatedly. Once they are able to memorize the instruction, they will not need to decipher the structure of the instruction but only need to retrieve from long term memory the components of the instruction in the right order to form subgoals that match the structure of the menu.

4.4   The Experimental Details

The Subjects

Sixty four first year computer science students with no previous experience in spreadsheet applications were recruited from the National University of Singapore. The incentives for participation were the opportunity to learn spreadsheet skills and also an equivalent of US $25 paid after the experiment. Computer experience of the subjects vary but they were recorded. Subjects were randomly divided into 4 groups of 16 students each.

The Physical Setup

The experiment was conducted in a computer lab with 16 IBM ATs equipped with color monitors and extended memory. The computers were on four rows of tables with four computers in each row.

The Software Setup

The 'RT-Capture' keystroke capture program developed at the University of Michigan was used to time stamp each keystroke. The key capture program has a resolution of two milliseconds. We used the Lotus macro language to modify the menu displayed and to present the task instructions on screen. Also, we had to remap some keys, e.g., '/' had to be remapped to a macro-call to display the desired modified menu for the treatment condition rather than to invoke the real Lotus menu.

The Dependent Measures

The dependent measures investigated are (a) the number of tasks executed wrongly, (b) the number of tasks with wrong search path leading to backtracking, (c) the number of tasks using cursor keys to issue the command instead of using the initial letters of the menu items, and (d) time per keystroke to issue the command.

(a) The number of tasks executed wrongly will indicate how the menu and instruction structures affect subjects' learning and error rate. A task executed wrongly is one where the outcome is not as described in the instructions; e.g., the whole spreadsheet is erased instead of a block of cells.

(b) The number of tasks with backtracking is another measurement of how the menu and instruction structures affect the subjects' learning and performance time. The need to backtrack is an indication of subjects having trouble forming the right subgoal or finding a menu item that matches the current subgoal.

(c) The number of tasks using cursor keys to issue the command is another measurement of subjects' learning. Subjects who form the right subgoal structure can use the initial letters of the menu items more readily to issue the commands instead of using the cursor keys to explore around the menu. We will however not penalize subjects for using the <return> key to invoke the first menu item on any level. Only when the right

and left cursor keys are used to get to the command item will the task be counted as one where the subject uses cursor keys to issue the command.

(d) The time per keystroke to issue the command is the total time to traverse the menu divided by the number of ideal keystrokes required. For example, the number of keystrokes to issue an adjust column width command in the old menu is four (/WCS) but three in the new menu (/CS). We exclude any keystrokes to specify arguments like column width or format type. etc.

The Independent Variables

The experiment design was a 2x2 between subject design with **menu version** and **instruction version** as the independent variables. Subjects were randomly assigned to one of the four treatments.

Menu: the Old inconsistent menu versus the New OA restructured menu.

Instruction: the Object-Action (OA) instruction versus the Action-Object (AO) Instruction.

Tasks

On each day, subjects had to perform the 10 different spreadsheet modification tasks described in Section 4.1 on each of the three different spreadsheets given for a total of 30 tasks. The tasks were randomly ordered in each spreadsheet. Subjects had to perform each task type, e.g., set global column width, three times, one for each spreadsheet. The order, parameters, and locations of each task type changed across the three distinct spreadsheets.

All task instructions were given online; there were no written instructions. Subjects were shown the first spreadsheet on screen when they began. The spreadsheets were small enough to fit on the screen. A set of labeled keys was used to invoke the

instruction screen and to get back to the spreadsheet to perform the task. For each task, subjects had to press the "read" key to read the next instruction. The instruction for the next task would replace the spreadsheet on the screen. After reading the instruction, subjects had to press a "do" key to get back to the spreadsheet to perform the modification task. The cursor would be placed on the cell where the modification task was. Subjects did not have to navigate to the cell where the modification was. With the time stamp on the special keys for reading instruction and starting the performance coupled with no navigation time, we know exactly how much time subjects spent in reading the instruction and how much time they spent in planning before the first keystroke. After subjects had performed 10 tasks on each spreadsheet, the next spreadsheet was brought up automatically by the macro program written to control the task sequence.

The Experimental Procedure

To accommodate the 64 subjects in the laboratory with only 16 computers, we divided the subjects into 4 batches of 16 students each. The experiment was a longitudinal study and each batch had to attend a total of four sessions. The 16 subjects in each batch were divided into 4 groups of 4 subjects each for the four conditions in the experiment. Each group of four subjects occupied one row of computers and received one of the four treatments. The treatment condition for each student remained the same throughout the four sessions. We chose not to run all 16 subjects in each batch under the same treatment condition as that may introduce self-selection and instruction biases. The treatment conditions for the rows of computers were rotated with each new batch of 16 subjects to counterbalance biases due to sitting position or the computer used in the lab.

Each batch of students had to attend four 2-hour sessions scheduled at one session every two days. Day 0^2 is on a Thursday, 4-6pm; Day 1, 2, and 3 are on either Monday, Wednesday and Friday, or Tuesday, Thursday and Saturday 3-5pm depending on the grouping.

Day Zero was a lecture where the experimenter introduced the basic spreadsheet concepts in the classroom. Subjects were taught the concepts of cell, row, column, label, entering cell contents, addressing, formulas, navigation within a spreadsheet, the mechanism to issue commands through the menu, etc. Since subjects from all four treatment conditions were present during the lecture, we used only a skeleton menu with item structure common to both the new and old menus for demonstration purpose. Only the items File-Retrieve and Block-Format were present in the skeleton menu, the other items were blanked out as 'XXX' on the menu. However, the function and meaning of each menu item, but not the structure, was explained clearly to the subjects.

Day One began with a 30 minutes guided hands-on exploration of the features of Lotus followed by a walk through of the skeleton menu. The function and meaning of each menu item were again explained to the subjects. Subjects were given five practice trials to familiarize themselves with the task sequence of invoking instructions on the screen and pressing a key to return to the spreadsheets to perform the task. Subjects were then left to complete the 30 tasks on their own.

Day Two began with subjects performing the same 30 spreadsheet modification tasks. They were encouraged to use the initial letters of the menu items to invoke the command instead of using cursor keys and the <return> key to invoke the command. Part

---

[2]We call the first day Day 0 as there was no lab session. Day 0 is just a lecture to introduce the spreadsheet concepts. Day 0 had no data. Day 1, 2, and 3 had actual lab sessions to collect data.

two of day two was a lecture on more advanced spreadsheet concepts followed by subjects entering a simple spreadsheet from scratch.

Day Three again began with subjects performing the same 30 spreadsheet tasks. They were asked to complete the same 30 tasks again after the first round. Thus there were two complete replications of the trials on the last day.

4.5   The Hypotheses

The inconsistent old Lotus menu does not allow users to form a consistent subgoal structure to learn and explore the menu. Subjects using the new menu can form a consistent <object><action> subgoal structure to traverse the menu. They can always form the subgoal of looking for the object of interest first then the subgoal of looking for the action to be performed on the object. For the main effect of *Menu*, we thus expect subjects given the new menu to perform better than those given the old menu in all four dependent measures. We state the hypotheses below in their alternate form only.

H1.   Subjects given the new <O><A> Lotus menu will have fewer errors than those given the old menu.

H2.   Subjects given the new <O><A> Lotus menu will have fewer tasks involving backtracking than those given the old menu.

H3.   Subjects given the new <O><A> Lotus menu will have fewer tasks using cursor keys to issue the commands than those given the old menu.

H4.   Subjects given the new <O><A> Lotus menu will have shorter time per keystroke than those given the old menu.

We do not expect any main effect for the independent variable *Instructions* since the OA instruction will help the subjects given the new menu but may hamper those given the old menu. However, we do expect a simple effect of the instruction on the new

menu users. Since commands in the old lotus menu can be classified into a few categories depending on their structure, we need to specify how the instruction format will affect each of these categories of tasks. We will do this for the analysis for the time per keystroke. In general, we are looking for the interaction effects between the menu structure and instruction structure to look like that depicted in Figure 4.5.

Figure 4.5 The Hypothesized Interaction Effects between Menu and Instruction
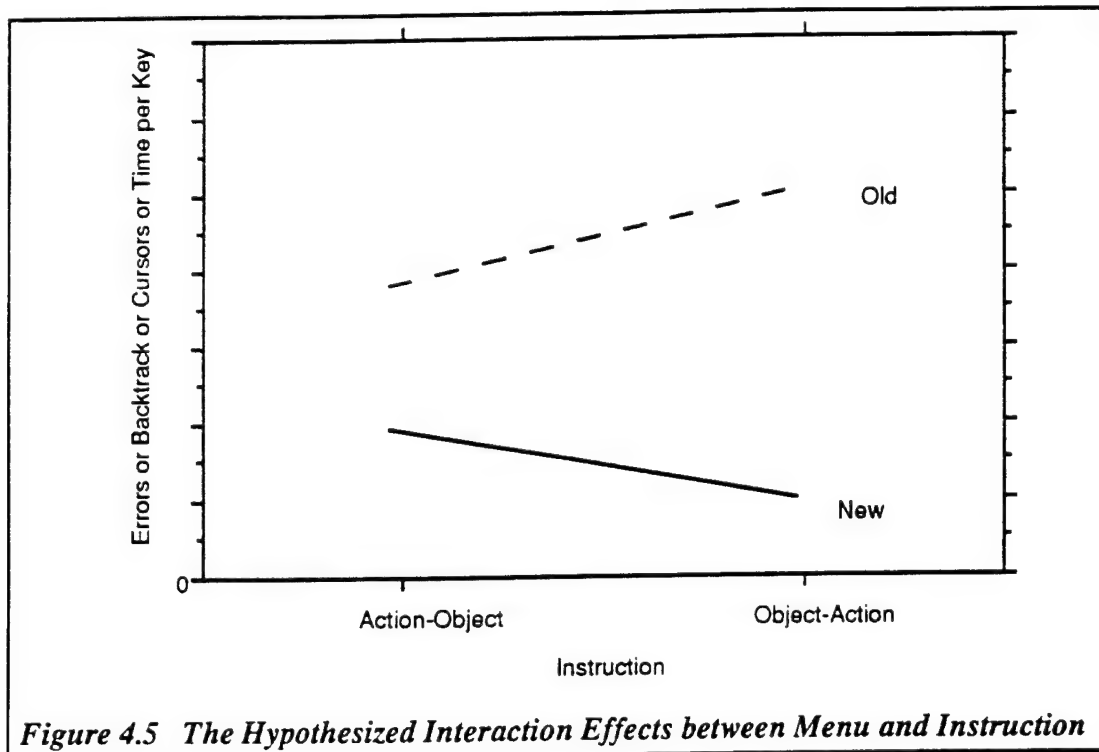
We expect that the new menu will always allow better performance than the old. However, we expect that the OA instruction will further help the performance of subjects given the new menu which has an OA structure. That is, we expect a main menu effect and a simple instruction effect on the new menu users. It is difficult to predict the slope of the old menu line due to the old menu's inconsistent structure. One way to overcome this difficulty is to analyze by individual task type. The slope of the old menu line can be positive as shown, or downward sloping depending on its task menu structure for the task investigated. If the particular task has a AO menu structure, the slope of the old menu

line will be as shown. If the task has an OA menu structure, then the slope of the Old menu line will be negative like that of the New menu line shown.
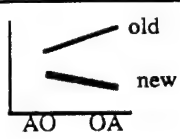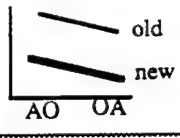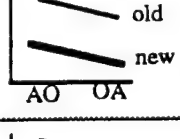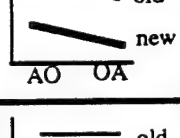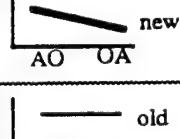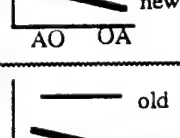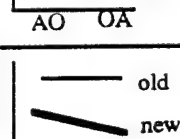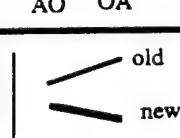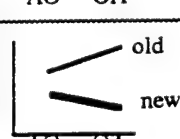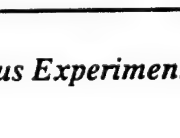
To analyze the number of errors, the amount of backtracking, and the number of tasks using cursors, we will count the occurrences in each task type. Since all the task types are grouped together in the ANOVA, we cannot predict the slope of the old menu line. The hypothesis for the interaction effect for menu and instructions on the first three dependent variables is stated below.

H5.   There are significant simple effects of the instruction structure on the new menu users for the three dependent measures: error rate, number of trials with backtracking, and number of trials using cursors to issue commands. The direction of the interaction is shown in Figure 4.5 but the slope of the Old menu line is undetermined.

Hypotheses Regarding Time Per Keystroke for Different Task Types

For the dependent measure of time per keystroke, we will analyze each task type separately. We can then predict the slope of the old menu line for the time per keystroke for each task type according to its old menu command structure. Performance of subjects given the old menu will depend on whether the instruction structure matches the menu structure for the particular task. Table 4.1 gives the expected outcome of the main effects and interaction effects by grouping the tasks according to the old menu structure.

For the main effect of the menu structure on the time per keystroke (fourth column in Table 4.1), we always expect the new menu to allow a shorter time per key compared to the old menu regardless of task types. Since the new menu has a consistent OA structure, we expect to see a simple effect that the OA instruction will always help the users given the new menu to reduce the time per key compared to those given the AO instruction and the new menu. This is due to our hypothesis that subjects given matching menu and instruction structure form the best subgoal structure without any extraneous

| Task | Command Keystrokes | | The Treatment Effects (Time per Key in Seconds) | | |
	Old Menu	New Menu	Menu	Instruction	Menu x Instruction
Copy/ Move Cells	/c	/bc	New better	Even	old / new (AO OA)
Erase Cells	/be	/be	New better	OA better	old / new (AO OA)
Format Cells	/bf	/bf	New better	OA better	old / new (AO OA)
Justify Cells	/bl	/ba	New better	OA better	old / new (AO OA)
Format All Cells	/wgf	/gf	New better	OA better?	old / new (AO OA)
Justify All Cells	/wgl	/ga	New better	OA better?	old / new (AO OA)
Set All Column Width	/wgc	/gs	New better	OA better?	old / new (AO OA)
Set One Column Width	/wcs	/cs	New better	OA better?	old / new (AO OA)
Row/ Column Delete	/wdr	/rd	New better	Even	old / new (AO OA)
Row/ Column Insert	/wir	/ri	New better	Even	old / new (AO OA)

Table 4.1   Hypotheses for the Individual Tasks in the Lotus Experiment

subgoals to explore the menu. Table 4.1 shows this interaction effect in the sixth column as a downward sloping line for the lower new menu line (the darker line).

We generally do not expect a main effect of the instruction structure on the time per keystroke (fifth column in Table 4.1). The effect of instruction will cancel since the old menu and new menu have different structures. We only expect to see the main effect of the OA instruction to be significantly better than the AO instruction for the group of 'BLOCK' tasks where both the new and old menu has the same OA structure. For the third group of "worksheet" tasks in Table 4.1, it is difficult to tell whether the OA instructions will be better. This group of tasks either has an OOA or an OOO menu structure. If we can treat these as having OA structure, then OA instruction will be better.

We do not generally expect to see an interaction effect of menu and instruction on time per key (see Table 4.1, last column). We expect that the OA instructions will help the new OA menu users even further. The bold horizontal line in Table 4.1 separates the task types into four groups according to the task's old menu structure. Copy/Move is by itself as it is the only one that has an "Action" single level structure. We expect that the AO instruction structure will aid the old menu users. The next category consists of commands dealing with a block of cells. Both the new and the old menu have the same OA structure. We expect that the OA instruction will aid the old menu users. The next category of tasks has an ambiguous old menu structure. The structure is either OOA or OOO. We hypothesize that the instruction will have no effect on the old menu users. The last group of tasks has an OAO old menu structure. If subjects ignore the "worksheet" item, then the old menu has an AO structure. We will then hypothesize that the AO instruction will aid the old menu users more than those old menu users given the OA instruction.

4.6   The Results of the Lotus Menu Traversal Experiment

Table 4.2 shows the distribution of subjects across the four treatments. Due to loss of data through computer disc crashes, only 57 out of the 64 subjects' data are usable for analyses. There are no significant differences among the four treatment groups in terms of attrition rate, sex, average years of computer experience. and the number of computer novices in the group.

	Menu/Instruction			
	New/AO	New/OA	Old/AO	Old/OA
Total Number of subjects	14	15	14	14
Sex of Subjects	5M 9F	4M 11F	7M 7F	10M 4F
Ave Yrs of Computer Experience	1.57	1.92	1.86	1.71
Number of Computer Novices	4	3	4	5

Table 4.2   Distribution of Subjects for the Lotus Experiment
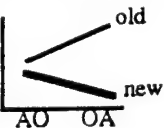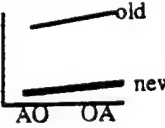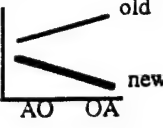
The results generally turned out to be as predicted by our hypotheses. We will describe the results in the following two sections. The first three dependent variables, the number of errors, tasks with backtracking, and tasks using cursor keys, are discussed in the first section. The time per keystroke according to task types follows.

4.6.1   The Number of Errors, Tasks with Backtracking, and Tasks Using Cursor Keys

Table 4.3 depicts the results of the first three dependent measures. As expected the new menu allows better performance on all three measures. Two out of the three instruction main effects are not significant as expected. Two out of the three interaction

effects turned out to be as expected. The unexpected results are marked with an ✖ in Table 4.3.

The Independent Factors	The Dependent Measures		
	Number of Errors	Number of Backtracking	Number that use Cursor Keys
Menu	New = .700 Old =1.800 p=.0219*	New =8.250 Old =22.80 p=.0009*	New =7.900 Old =36.700 p=.0001*
Instruction	OA=1.550 AO=.950 p=.2923	OA=16.200 AO=14.850 p=.2077	OA=24.850 ✖ AO=19.750 p=.0011*
Menu x Instructions	old / new (AO OA) p=.0304*	old / new (AO OA) ✖ p=.8649	old \ new (AO OA) p=.0001*

Table 4.3   *The Effects of Instruction and Menu on the Number of Errors, the Number of Tasks with Backtracking and the Number of Tasks Using Cursor Keys*

Number of Errors (Tasks Executed Wrongly)

There were not many errors after the first day as subjects had generally learned to execute the tasks after the first day. Subjects using the new menu had fewer errors than subjects using the old menu. A simple effect test shows that the "OA" instruction hampered the old menu subjects' performance [F(1,9)=7.945,p=.0201]as six "block erase" tasks were executed as "worksheet erase" tasks. Also, seven "block format" tasks of "OLD/OA" subjects were executed as a "global format" task. These two sets of errors accounted for 50% of the "Old/OA" errors. These errors occur because subjects when given the "OA" instruction, formulated a subgoal of finding the possible object on the menu and then the action. They thought that *Worksheet* possibly fits the description of a

block of cells and this triggered them to wander down the wrong path. Since *Erase* was at the second level menu of *Worksheet*, there is no indication to the subjects that they had traversed down the wrong menu tree.

Menu	Instruction	
	Object-Action	Action-Object
New	0	1
Old	8	5

Table 4.4   Number of /WE Attempts in the Erase Block Command

Table 4.4 shows the number of attempts to issue the "worksheet erase" command in the four treatment groups. Subjects given the new menu did not encounter such problems as there was no confusing "worksheet" command at the first level menu. Only one subject given the new menu attempted the "global erase" command. Subjects given the old menu attempted to issue the "worksheet erase" command more frequently. This phenomenon illustrates the label-following heuristic brought up by Polson and Lewis (1990). Subjects attempt to execute commands with labels that resembles the description of the task. The opportunity for errors was even greater when subjects formed the subgoal of looking for an object first and found that "worksheet" matched that description at the first level.

Number of Tasks with Backtracking

The new menu significantly reduces the number of tasks with backtracking. This confirms that subjects who could form a consistent subgoal structure were able to learn the menu system faster. The mean number of tasks with backtracking is out of a possibility of 126 trials (14 subjects[3] x 3 sessions x 3 tasks per task type). To allow for

---

[3]Results adjusted for New/OA which has 15 subjects.

learning, the Day One data are excluded in the analyses. We expect subjects to spend most of Day One exploring the menu so backtracking will be rampant. However, even an analysis of Day One data shows that the new menu reduces backtracking [New=7.2[4], Old=17.2, $F(1,9)=81.81$, p=.0001]. Therefore, excluding Day One results will not bias our results.

As expected, the main effect of Instruction is not significant. However, neither the interaction effect of Menu and Instruction nor the simple effects of instructions are significant which is contrary to expectations. This can be explained as there were more typing errors for the "New/OA" subjects since they could use the initial letters of the menu items to issue commands (see discussion in next section). Typing such letters led to more opportunity for slips than those simply using cursor and enter keys to issue the command.

Number of Trials Using Cursor Keys

We again exclude Day One data from the analyses as we expect subjects to have to explore more during Day One. The new menu significantly reduces the number of trials using cursor and enter keys to issue commands. The interaction effect between the menu and instruction structures on the number of trials using cursor keys is also significant as expected. The "New/OA" subjects' performance (an average of 2.10 trials[5]) is better than the "New/AO" subjects' performance (13.70 trials) [$F(1,9)=377.84$, p=.0001]. The results show that subjects given the consistent new menu could quickly learn the menu structure. They could form consistent subgoals and thus could use the initial letters of the menu items to issue the commands more easily. More subjects (36.70

---

[4]Out of a possible 42 tasks (14 subjects x 1 session x 3 tasks).

[5]Out of a possible 126 trials.

trials) using the old menu resorted to using cursor keys to issue commands since they could not form a consistent subgoal structure. It was easier for them to use the cursor keys and to look for the menu item that matches one of the subgoals without worrying about the order of the subgoals. Furthermore, subjects given the consistent menu with the matching "OA" instruction performed best since they could form the right subgoal structure as the subgoals were in the order that were conceived in the task space $F(1,9)=106.98$, p=.0001]. They did not need extra subgoals to buffer the out of sequence task action subgoals.

The results show that the keystroke level model's assumption that task execution is the same no matter how the task is acquired is not valid here. Subjects acquiring the task in a way that allows them to form subgoals that are consistent with how they think of the task, perform better than when the subgoal mismatches the task structure.

The significant effect of instruction on the number of trials using cursor keys is not expected. This effect can be accounted for by the large number of tasks involving cursor keys for the "Old/OA" subjects. The object first instruction was more confusing for subjects using the old menu as the subjects could not learn the menu structure due to the inconsistent menu organization. For the tasks that has an "AO" old menu structure (e.g., copy, delete row, etc.), the "OA" instructions caused more errors due to the need to remap the subgoals. The frequent errors led to slow learning thus subjects needed to use the cursor keys to issue commands. For the tasks that had an "OA" structure in the old menu, subjects often were led down the wrong path. There were many objects on the first level that may suit the description of the objects mentioned in the instructions. The item "worksheet" is especially confusing since subjects often substituted it for block tasks. This frequent wrong path again led to slow learning making the use of cursor key rampant for the "Old/OA" subjects.

Number of Trials with Wrong Command in Move

There is an incidental result that gives further evidence that subjects were trying to form a consistent subgoal structure to traverse the menu. The evidence came from the "Move[6] Block" command. In the new menu, the command is "/ Block Move", Table 4.5 shows the number of subjects given the new menu that attempted to use the command by specifying "/ Move" and the reverse scenario described below. Since most subjects given the New menu use the initial letters to issue the command, we can attribute some of these errors to slips in typing.

Menu	Instruction	
	Object-Action	Action-Object
New	5	3
Old	11	10

Table 4.5   Number of Slips in the Copy/Move Command

The interesting result is from the reverse scenario. Subjects given the old Menu had an unreasonably high frequency of trying to use the "/ Block Move" command instead. This reverse mistake cannot be attributed to slips as the subjects had to insert intentionally the "Block" command before the Move command. Furthermore, since most subjects given the old menu used cursor keys to issue commands, they had to use the cursor key to move to the Block command first (not the first item on the first level) before they could select the "Block" item.

This mistake of trying to use "Block Move" instead of the simple "Move" command in the old menu can be attributed to subjects trying to form a consistent

---

[6]or Copy

<object><action> subgoal structure to traverse the old menu. This shows that the saving of one keystroke for the move command by moving it up one level is not significant considering the penalty of possible errors and slow learning.

4.6.2 The Time Per Keystroke

We now describe the results for the time per keystroke by individual task types. We first describe the overall progression in time over the four sessions before we describe the time per key for the individual task types.

The Overall Time for the Four Sessions

Figure 4.6 shows the average time per task (excluding time to read instruction and specify command arguments) over the four sessions. Tasks performed during Day One took a much longer time as subjects had just started learning the spreadsheet application, the menu structure and the task concepts. There was a lot of menu exploration and backtracking. There was a big improvement from Day One to Day Two as subjects had at least three chances to look at each task type during Day One. For subjects using the same menu, Day Three performance almost reached asymptote as there was little difference between the task times for subjects session 3a and session 3b.

From the task time, we see that the new menu allows shorter performance time from Day 1 onwards. The effect of menu on task time is significant [$F(9,27)=14.18$, $p=.001$]. For the old menu, as expected the effect of instruction on task time is not significant since individual tasks have different command structures. For the new menu, the OA instruction allows slightly better performance in the first three sessions but the effect is not significant. In session 3b, the second time that subjects performed the 30 tasks during the last day, the OA instruction did not help the new menu users. This can be explained as by then, subjects had memorized the instructions in the desired structure

that matches the menu structure. The structure of the instructions was no longer important.

Figure 4.6 *The Time Per Task Over the Four Sessions for the Different Treatments*
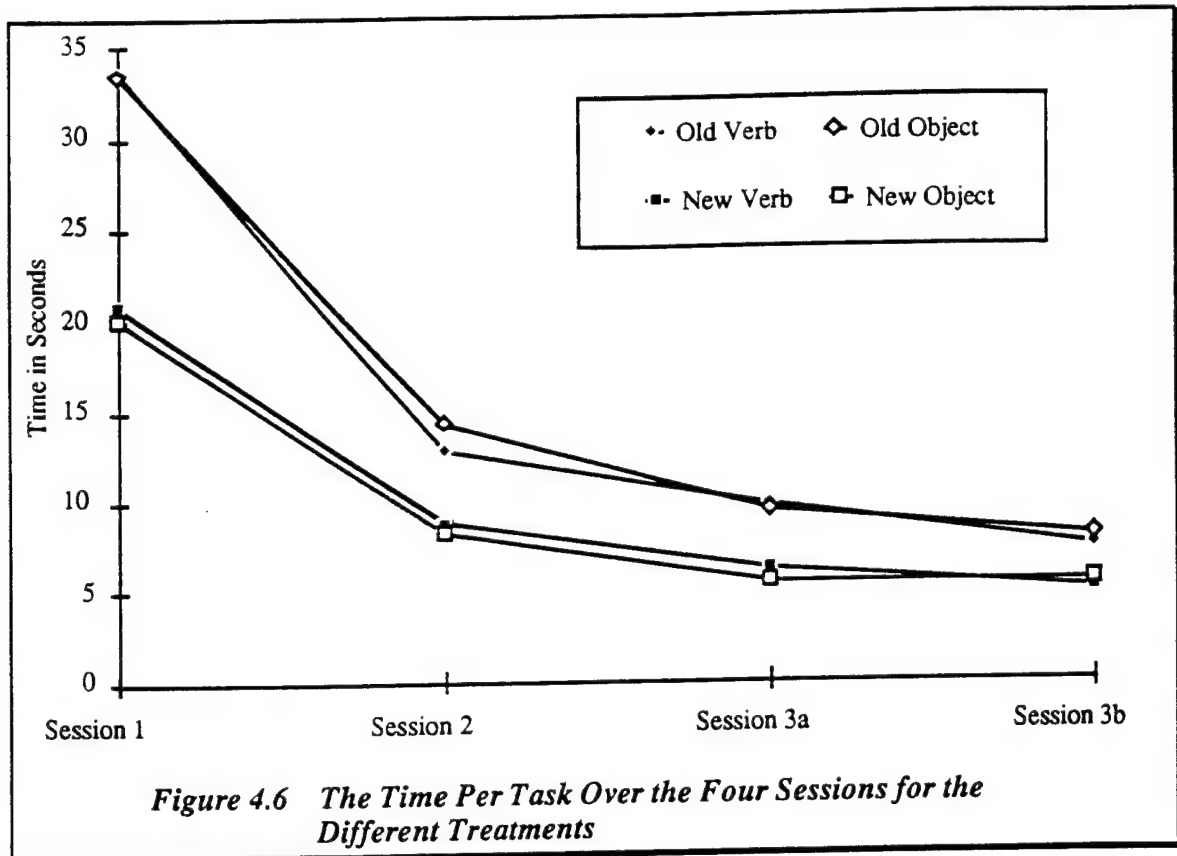
The Effects of the Independent Variables on Time Per Key for the 10 Different Tasks

Table 4.6 shows the ANOVA results for the time per key for the 10 different task types. The results are generally as stated in the hypotheses in Table 4.1; but there are a few exceptions for which we will provide plausible explanations. Results that are unexpected are marked with "✖" and will be explained in the discussion.

Menu. The "Menu Structure" has a significant effect on the time per key for eight out of the 10 task types. This shows that subjects given the new menu were able to form a consistent and efficient subgoal structure to traverse the menu. The "New" menu reduces time per key for all 10 task types but only eight results are significant.
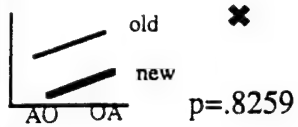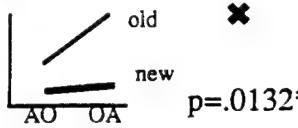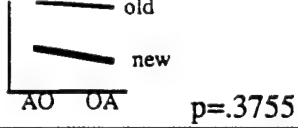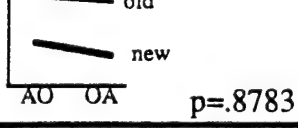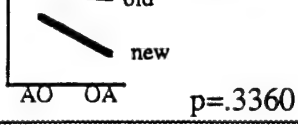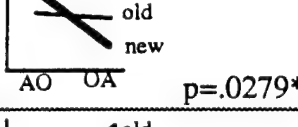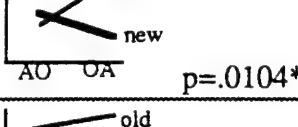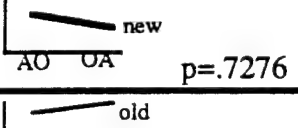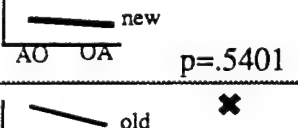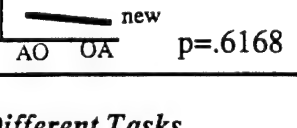
| Task | Command Keystrokes | | The Treatment Effects (Time per Key in Seconds) | | |
	Old Menu	New Menu	Menu	Instruction	Menu x Instruction
Copy/ Move Cells	/c	/bc	New=2.101 Old=2.972 p=.0001*	OA=2.696 AO=2.355 p=.876	✖ old / new p=.8259
Erase Cells	/be	/be	New=1.953 Old=3.257 p=.0001*	OA=2.919 AO=2.257 p=.0032* ✖	✖ old / new p=.0132*
Format Cells	/bf	/bf	New=3.157 Old=3.874 p=.0080*	OA=3.436 AO=3.585 p=.6039 ✖	old new p=.3755
Justify Cells	/bl	/ba	New=2.831 Old=4.355 p=.0001*	OA=3.435 AO=3.707 p=.4976 ✖	old new p=.8783
Format All Cells	/wgf	/gf	New=1.777 Old=2.203 p=.0220*	OA=1.866 AO=2.111 p=.1919	old new p=.3360
Justify All Cells	/wg:	/ga	New=1.998 Old=2.130 p=.4347 ✖	OA=1.829 AO=2.308 p=.0028*	old new p=.0279*
Set All Column Width	/wgc	/gs	New=1.979 Old=2.397 p=.1199 ✖	OA=2.280 AO=2.086 p=.4148	old new p=.0104*
Set One Column Width	/wcs	/cs	New=2.668 Old=3.318 p=.0352*	OA=2.973 AO=3.002 p=.9584	old new p=.7276
Row/ Column Delete	/wdr	/rd	New=1.744 Old=2.829 p=.0001*	OA=2.297 AO=2.256 p=.7406	old new p=.5401
Row/ Column Insert	/wir	/ri	New=1.609 Old=2.116 p=.0009*	OA=1.775 AO=1.944 p=.2631	✖ old new p=.6168

Table 4.6   The Results of the Time Per Keystroke for the Different Tasks

For "Justify all cells" and "set all column width", the results are in the predicted direction but not significant. We can explain this as, since "worksheet" is the first item on the first level menu, subjects given the old menu often resorted to using the return key to select the "worksheet" item. Thus, they engaged in a rule-based behavior to hit the '/' key followed by the 'return' key for any worksheet command. Since hitting the <return> key is considerably faster than hitting any letter key, the extra <return> key time is negligible. This will help the subjects given the old menu since we are considering time per key, not the total time to issue the command. However, from the results in Table 4.6, we can still see that for the two other "worksheet" tasks, "format all cells" and "set one column width," the subjects given the new menu still performed significantly better than the subjects given the old menu. Also, we need to consider the previous results on the number of erroneous trials and trials using cursor keys that showed that subjects given the old menu had difficulty learning the menu system on the whole.

Instruction. As expected, the "Instruction" structure by itself does not significantly affect the time per key. Even for the "block" task where the new and old menus have the same <object><action> structure, the "OA" instruction did not help. Since the old menu lacks a consistent structure taken as a whole, subjects could not take advantage of a few tasks that have consistent instruction-menu structure. Since subjects given the old menu lack a consistent subgoal structure to explore and traverse the menu, the instruction structure will not matter much to them.

The unexpected reversal in time per key for the "Erase Cells" task where the "OA" subjects have a higher time per key than the "AO" subjects can be explained easily. The higher time per key for the "OA" subjects is due to subjects given the old menu trying to issue the "Worksheet Erase" command instead, causing backtracking (see discussion in previous section).

Menu x Instruction.   The directions of the interaction effect sof menu and instruction on the time per key (Column 6 in Figure 4.6) generally turned out as hypothesized in Table 4.1. However, only three out of the ten interaction effects are significant and only one (Justify all cells, p=.0003) out of the ten simple effects of instruction on the new menu users is significant. This indicates that instruction is possibly a weaker manipulation of how users think of the task. Since subjects saw the same instructions repeatedly, the instruction structure may not matter after subjects had seen the instruction a few times. They may have memorized the instruction in the structure most desirable to them, i.e., <O><A> or <A><O>. However, if we again consider the results discussed in the previous section on the number of erroneous tasks, the number of tasks with backtracking, and the number of tasks using cursor keys, the instruction format does have an effect on the subjects' ability to learn and traverse the menu system.

Seven out of the ten menu and instruction interaction effects are in the direction predicted in Table 4.1. One outlier, the "Erase Cells" task, can be explained easily. Subjects given the old menu performed poorly when given the OA menu as they were trying to use the "worksheet erase" command instead (see discussion above). This causes the old menu line to slope upwards instead of downwards as predicted.

There are no good explanations for the other two outliers. For the "Copy/Move" task, the "OA" instruction unexpectedly did not help the new menu users. This may be attributed to the wordings in the instructions. For "Copy/Move" instructions, we did not describe the task directly as "For the Block of cells, Move them to..." but rather, for example, "For the labels in the column headings, Move them to...". This may have caused users not to associate "Copy/Move" task as a "block" command for the "New" menu users.

For the last task in Table 4.6, the slope of the old menu line is not as expected. However, we mentioned in the section on hypotheses that the prediction depends on how subjects view the structure of the old menu for the "Insert Row" task. The old menu structure for this task has an <O><A><O> structure and the prediction can be either way depending on whether subjects considered the "worksheet" item as an object.

In conclusion, we do find that the interaction effects are generally as predicted in the hypotheses. The results in this section together with results from the previous sections show that consistent menu structure allows users to form a consistent subgoal structure to learn and traverse the menu. A matching instruction structure will further help users traverse the menu by avoiding the use of extra subgoals to buffer the out of sequence action subgoals.

4.7   Discussion of Results

The results of the Lotus Menu traversal experiment again helps to illustrate the subgoal theory. When the interface does not allow users to execute subgoals in the sequence conceived in the task space, learning and performance are hampered. This again shows that the KLM assumption of independence between task acquisition and task execution is not valid in this situation. The instruction format dictates how users acquire the task and the menu format dictates how users can execute this task. When there is a task-system mismatch, users need extra subgoals to buffer the out of order subgoals formulated in the task space thus affecting the performance.

This experiment again illustrates why a simple qualitative difference in the interface design can lead to big differences in performance although the number of keystrokes remains the same. We again use the subgoal structure to explain the performance differences. It is whether the interface allows the execution of the subgoal structure formed in the task space that is important rather than the final keystroke count.

4.8   Conclusion

This chapter describes yet another experiment to illustrate how subgoals are used to bridge the gap between the task and system space. In Chapter 3, we described the formula editor experiment where the task and system space operate on different objects. Here, we describe a Lotus menu traversal experiment where the task and system space operate on different sequences of action subgoals. Again, subgoals extraneous to the task need to be created to bridge the task-system gap. These two experiments taken together illustrate the importance of making the mechanisms of the system match the thoughts and goals of the user. When there is a task-system match in terms of the object structure manipulated or the order of action sequence allowed, there is a higher likelihood that users can learn to use the system faster and progress to rule-based or skill-based behaviors. Our theory and experiment explain that these performance improvement gains are due to the elimination of the planning, monitoring, and validating of extraneous subgoals.

CHAPTER 5

GENERAL DISCUSSION AND A DESIGN METHOD

In this chapter, we will discuss the contributions and implications of this research, and the ways this research may be extended in the future. To conclude this thesis, we present a design method for designing better interfaces based on the principles learned from this research.

5.1   Contributions of This Research

This research contributes to the cumulative HCI research literature and in particular, contributes to the cognitive modeling approach. It provides a comprehensive framework to unify the fragmented HCI theories and cognitive models. The subgoal theory extends the current GOMS models by addressing non-optimal behavior and demonstrating sources of some performance difficulties. The two empirical studies and their results illustrate design principles that can help designers to create better interfaces that eliminate such performance difficulties. We will present the design principles as a design method at the end of this chapter. We elaborate on some of these contributions below.

Develops an HCI Cognitive Modeling Framework

This research develops a conceptual HCI framework that integrates important theories in the HCI and human factors literature. The framework is *integrative* as it provides a classification scheme to help researchers structure HCI literature and investigate the stages of activities and cognitive processes underlying HCI activities. The

literature review in Chapter Two illustrates how the framework is used to integrate and view the different HCI cognitive models as addressing different facets of the same design issues. The framework is integrative also in providing means for researchers and designers to draw on other relevant human behavior literature to address HCI issues. Furthermore, the framework is *analytical* as it points out areas that need further research by looking at which areas of the framework are not addressed by current HCI theories. The framework is also *prescriptive* as it brings insights to designers since it identifies bottlenecks and highlights critical design issues in the HCI activity cycle. This framework thus has important implications for the research and design of computer interfaces.

Models the Cognitive Mechanism that underlies Task-System Mismatch

This research investigates how people plan and execute tasks, and in particular how subgoals are formed during computer supported activities. This research proposes a subgoal theory that task-system mismatch will lead the users to create extraneous subgoals that are difficult to form, monitor, and validate. This cognitive mechanism of creating extra subgoals contributes to making an interface difficult to use. For example, we illustrate using the formula editor experiment how users had to create extra subgoals when the task and system space operate on different object structures.

Extends Current GOMS Models

This research suggests that we should investigate knowledge-based activities (those involving problem solving) where the subgoals play an important role. This research suggests ways that the GOMS model can be extended to address non-optimal behavior by examining the subgoal structures. This research extends the state of the art of the KLM by modeling the behavioral effects when two interfaces generate about the same number of keystrokes but require very different subgoal structures for task

performance. This research extends the GOMS model to address the mappings from the task space to the system space. Current GOMS models and the extensions only address the system space through the modeling of the operations in the system space. This research addresses how performance can be affected by the way users decompose the task goal into subgoals as constrained by the system space.

The formula editor study shows that why certain interface will not allow skill-based behavior since the subgoal structures are too complicated, making users overextend their working memory that leads to erroneous and slow performance. The Lotus menu study show how we can extend the KLM to address novice learning behavior by examining the subgoal structures users employ to traverse the menu system. The two studies together illustrate that consistency within an interface (modeled by TAG [Payne & Green, 1986]) is important but consistency between how users think of the task with the way the interface allows the task to be executed (modeled by the subgoal theory) is also important.

Suggests Design Principles for Formula Editors and Hierarchical Menus

The empirical work show us how some redesign can drastically reduce the complexity of the interaction process. The formula editor experiment suggests that editors should work with the structure of the language it manipulates. This avoids the need for extra subgoals to overcome the unnatural mapping between the task language and the system language. The Lotus menu traversal experiment suggests that the internal consistency of a menu structure is important, but it is also important that the structure be consistent with how the users think of the tasks. This consistency between users' view of the task and the interface eliminates the need for users to create extra subgoals to overcome the unnatural mapping.

5.2   Implications of this Research

The HCI framework and the theory of the roles of subgoal formulation in bridging task-system mismatches have important implications for both researchers and practitioners.

The HCI Framework and Research

This research points out that it is premature to develop a comprehensive cognitive model as the underlying cognitive theories are fragmented and the resulting model may be too complicated for practical use. The HCI framework helps to overcome the need to have a comprehensive HCI performance theory. It helps researchers and designers understand the activities and the cognitive processes involved in HCI thus knowing how to draw on the related human behavior theories from different fields. The HCI framework points out the particular cognitive processes that need to be addressed at the various stages of HCI activities. Consequently, interface designers can use the framework to draw on different theories, HCI cognitive models, and empirical results to address the design issues currently under focus.

The Future Need for HCI Research

The HCI framework provides a means to review and structure the existing HCI empirical literature. For example, this research uses the framework to classify the Formal Grammar family of models as rule-based models as they only address that aspect of the framework. The framework helps researchers identify areas that need further research and suggest hypotheses for testing. For example, we use the framework in this research to review the HCI literature and find that there is a lack of research investigating knowledge-based activities and the feedback portion of the HCI framework. We then state the subgoal theory to address the knowledge-based behavior not modeled by current HCI models.

Some Questions Answered

By investigating how users form subgoals as a cognitive control mechanism for knowledge-based behavior, this research begins to answer questions brought up in the literature reviewed in Chapter 2. These answers are not yet complete but point to possible research directions that extend the subgoal theory to cover situations that current HCI models do not address. Examples of these questions and the answers that this research begins to provide are:

How do people move smoothly between skilled performance and problem solving?
> They use subgoals formulation as a cognitive control mechanism to move from problem solving behavior to skill-based behavior and vice versa. Subgoals have a diminished role in skill-based behavior but are essential in knowledge-based behavior.

How do we design for consistent user interfaces?
> We try to design an interface such that it is consistent with how the users think about the object structure and action sequence in the task space. This way, extraneous subgoals can be minimized and the system space made as transparent to the users as possible.

How do people produce and manage errors?
> People produce errors when there is a slip during skill-based behavior, a mistake during rule-based behavior, or a resource error during knowledge-based behavior. People manage errors by reverting to a higher level cognitive control and instigating a new subgoal to reverse the damage done by the error.

What contributes to mental workload?
> The planning, monitoring, and the validating of extraneous subgoals consume large amount of working memory capacity.

Which stages of the activities take the longest time?
> The planning, monitoring, and validating of extraneous subgoals take the longest time in the HCI cycle of activities. The number of keystrokes different interfaces require are not important unless the interfaces also lead users to form the same subgoal structure.

Implications of Findings for Interface Design

This research shows that GOMS in its current form is incomplete for use by system designers. An important part of designing a system is setting the tasks that users will perform, the sequence of objects to be selected and actions on those objects. The system designer needs to model how users will translate the external tasks to the internal subgoal sequence of actions upon the objects. The implications of the findings in this research for interface design are two fold. First, it illustrates that it is imperative that interfaces allow users to form subgoals that reflect the task structure as conceived in the user's task space. Second, it is also imperative that the users first be trained to decompose complex tasks into appropriate subgoals in the task space. That is, users need to have the appropriate domain knowledge before using the system. With this qualitative aspect of interface design solved, the quantitative predictions of HCI cognitive models can be made even more rigorous and accurate.

5.3   Possible Future Research

This research has some limitations reflecting deliberate choices intended to keep the research within scope and the discussions within reasonable length. We suggest some possible future research to address these shortcomings.

The HCI Framework

The HCI Framework attempts to address all relevant HCI issues making it a potentially comprehensive model. However, the price to pay may be unnecessary complexity and an over-simplification of some important underlying issues. Further research will need to address exactly which portion of the framework should be applied in which design situations. Ultimately, the framework can form the basis of an expert system for designers (e.g., [Barnard,Wilson, & Maclean, 1987]).

Furthermore, the framework should be extended to address how and when users engage in the different levels of behavior in the framework. The framework in its present form does not describe how the transition between the different levels of behavior occurs, merely that it occurs. We can draw on research on skill acquisition (e.g., [Newell & Rosenbloom, 1981]) and problem solving (e.g., [Newell & Simon, 1972]) to help us refine our framework and the design theories it can generate.

The framework also addresses only situations where the task goals are well defined. Future research may extend the framework to address situations where the task goals are not well defined, e.g., creative design situations. We can draw on the findings and theories in the design problem space research (e.g., [Goel & Pirolli, 1989; Reitman, 1965]).

The current framework also does not address how users formulate subgoals. There is a need to understand how users formulate subgoals to execute complicated tasks or to overcome a bad interface. Besides creating extra subgoals, users can sometimes restate the goals to suit the interface. We need to understand the different cognitive strategies during knowledge-based behavior for subgoals formulation. For example, the Soar [Laird,Newell, & Rosenbloom, 1987] cognitive architecture can provide the mechanisms to help us model how users formulate subgoals.

The Subgoal Theory

The subgoal theory in its present form is only a qualitative theory of how users employ subgoals to bridge the task-system space mismatches. It does not attempt to quantify how extra subgoals will affect the performance time; nor does it attempt to quantify how extra subgoals will contribute to the needs for planning and evaluation. Future research can extend in this direction. We will need to model explicitly the process of storing and retrieving subgoals from working memory, the scanning process to acquire

the task goal, the planning process to decompose the task goal, and the checking process to ensure that a subgoal is accomplished successfully.

The subgoal theory also does not address the feedback loop portion of the HCI framework. Future research can address how good feedback may reduce the need for planning and evaluation. For example, we can incorporate the structural display of formulas for the semantic editor and see how that will further help users reduce the needs for planning and evaluation.

We have not presented a design method that can be applied by designers to create an interface that avoids 'bad' subgoal structure. It is important that useful theory should lead to real world application. We will sketch a design method in the last section.

The Experiments

We can extend the empirical studies used in this research in many ways to generalize the results and to address other issues. We can extend the formula editor study to other functional languages. We can try to apply the design principle in creating the semantic editor to, for example, a LISP statement editor. We also can extend the formula editor study by addressing the feedback loop. We can see whether a structural display of the formulas for the semantic editor further improves subjects' performance; or how a linear editor with structural display may eliminate some planning and evaluation needs. The performance differences between subjects using the semantic editor with and without the structural display can shed light on the role that feedback plays.

We also can extend the Lotus menu study to other applications like database query. In database query, users also conceive a subgoal sequence in the task space to extract the information needed. Depending on how well the interface language supports this subgoal structure, users may have difficulties, e.g., forgetting the "JOIN" statement in a relational database query language [Smelcer, 1989].

In summary, this research provides a unifying explanation of separate results, and begins to develop a design method that can help designers create better interfaces. Specifically, these interfaces would allow subjects to form subgoal structures that reflects the task structure.

5.4 A Design Method for Creating Good Interfaces

We discussed in Chapter Two that the purpose of the interface is to support task performance. There are three components to how well a user can perform tasks with a system: (a) the user's task domain knowledge and system knowledge; (b) the concepts and structures represented in the system space; and (c) how well the interface bridges the task-system space. All three aspects can improve task performance. Here we have assumed that the user is a task domain expert and have not explored that aspect further. The user we have focussed on knows what the task goals are in our framework but may not know the methods to accomplish the task goals in that particular system context.

The first possible way to support task performance is to train the users on the system's methods, those system commands available that can achieve the task regardless of how the task goals are conceived in the task space. This is a brute force method to improve task performance by forcing users to adapt to the interface. This is certainly not an ideal solution, and may have insurmountable long term costs. In a previous longitudinal study (in preparation), we found subjects using Lotus spreadsheet still committed many errors at the end of the two year period. Also, in the formula editor study, we show that the linear editor imposed so much working memory load on the users making transition into skill-based behaviors impossible for the users.

The next alternative to improve task performance is to improve the target system language to match the concepts and structures in the task space. This is trying to bring the system space closer to the task space. For example, object-oriented programming

languages provide better concepts and structures for programming complex object relationships; a direct manipulation interface provides a direct mapping of the structures for two-dimensional drawing, etc. This is a much more desirable way to improve task performance. However, this route is not available if users are not willing to learn a new interface language, or the underlying technology is not viable for supporting a new language. For example, a text-based terminal will not support a direct manipulation interface.

The third alternative to improve task performance is to design a better interface to bridge the task space and the system space. The interface shields the underlying complexity of the system language from the users by providing tools to bridge the task-system space. Using our semantic editor as an example, although the target language is still linear text, the semantic editor effectively bridges the task-system space by manipulating hierarchical formula structures in the task space. Another example will be the reorganized Lotus menu system that presents a rather complicated system command structure to the users in a consistent and logical way where the items sought are found in the order they are sought.

In short, a good interface should focus on the task domain and make the "mechanisms of the system match the thoughts and goals of the user" [Hutchins,Hollan, & Norman, 1986]. We want to design interfaces that help users accomplish task goals without introducing any subgoals extraneous to the task goals. This is achieved by making the system as transparent as possible to the users. We outline below a design method based on this principle.

The following design method assumes that the designer has access to the users' task domain knowledge. The designer is either a task domain expert or has access to a task domain expert. The task domain expert knows all the task goals in the task domain.

Step 1.   List the Task Goals, Objects and Actions in the Task Space

Ask the task domain expert to list all possible task goals in the task space. We discussed what task goals are in Chapter Two where they are defined as the smallest task that can be couched in the task domain language. For example, in the spreadsheet domain, a task goal is to 'insert a column.' All possible *objects* and *actions* of interest related to the task goals in the task space are also solicited from the task domain expert. For example, objects in the spreadsheet domains are *column, row, cell*, etc.; actions in the spreadsheet domain are *insert, delete, copy, move*, etc.

Step 2.   List the Objects and Actions in the System Space

The designer will list all the *Objects* and *Actions* s/he intends to include in the system space. The designer next evaluates whether these objects and actions are the same as those listed by the task domain expert; whether they refer to the same object structures and concepts in the task space. If not, the designer has to think how to bring the objects and actions in the system space to match those in the task space. Ideally, the objects and actions in the task and system space should be identical.

Step 3.   List the Methods to Accomplish Each Task Goal

The designer at this stage will list the methods provided by the interface to achieve each task goal. The designer at this juncture will avoid providing multiple methods for one task goal to avoid unnecessary complications. Each method will consist of a set of operations (actions upon objects) and/or subgoals to accomplish the task goal. The designer confirms with the task domain expert that the methods reflect the ways the expert will break the task goals into smaller subgoals; i.e., the subgoal structure reflects the task structure. If not, the designers will need to provide different operators that result in a subgoal structure that reflects the task structure.

Step 4. Evaluate the Method/Subgoal[1] Structure

The designer can then evaluate the methods/subgoals by coding them using for example the GOMS model notation. The purpose of this step is to evaluate the structures of the methods and subgoals. The GOMS notation will reveal how long the methods/subgoals are, how many steps each method contains, whether the steps have a consistent and logical structure, etc. The method/subgoal structures revealed can suggest many improvements to the interface. The designer thus will evaluate:

1. Are the methods too long?

   Methods should not contain more than five to seven steps. Long methods will create unnecessary memory load. Any methods that contain too long an action sequence must be broken down into smaller subgoals. Alternatively, the interface can provide higher order operators that accomplish multiple subgoals to shorten the method.

2. Do the methods require excessive planning or evaluation?

   Try to eliminate long planning or evaluation by making the method to accomplish the subgoal easy to acquire and evaluate. Methods that reflect the task structure should be easier to acquire and evaluate. The interface also should provide easily identifiable feedback that indicates when the methods are executed successfully.

3. Do the methods allow users to execute task actions in the sequence conceived in the task space?

   Methods must specify subgoals in a sequence that reflects the way users will decompose the task goal. If not, redefine the method.

---

[1]We explain in Chapter Two that methods are means to accomplish task goals and subgoals. We will use method and subgoal interchangeably in the following discussion.

4. Are the methods/subgoals distinct and independent?

The delimitation of each subgoal should be distinct so that it is easy to acquire. Subgoals should be independent in that one uncompleted subgoal should not generate another subgoal. If a subgoal triggers another subgoal, there should be some visual feedback to remind users that there is a subgoal not completed. The methods should not generate nested subgoals that quickly stack up and exceed memory load. Methods should have self-closure in that they do not generate subgoals recursively.

5. Are the subgoals/methods meaningful and complete and reflect the task space's objects and actions?

The subgoals should reflect the concepts and actions in the task space so that they are meaningful and not extraneous as a result of having to use a bad interface.

6. Do the methods/subgoals have consistent grammar and semantics?

There should be a higher order rule governing a family of methods/subgoals for a set of related task goals such that the subgoals have consistent syntax and semantics. Consistent syntactic structure will ensure subgoals have consistent ordering of "object" and "action" within each subgoal. Consistent semantic structure will ensure subgoals are executed in the sequence as conceived in the task space.

7. Do the methods/subgoals have only a few parameters?

Each subgoal should not have too many parameters that need to be specified. If so, the system must prompt the users for all parameters needed.

Step 5. *Evaluate the Interface Support for Subgoal Accomplishment*

The designer must next investigate how well the interface supports the execution of the subgoals.

1.   Does the interface allow rapid learning of the commands or actions to execute the subgoals?

The interface must make the repertoire of available commands or actions to accomplish the subgoals salient. Given a subgoal, the interface must allow a quick pruning of the search space so that the user can discover the command or actions to execute the subgoal quickly. There must be no confusing alternatives that may seem to accomplish the same subgoal but with totally different outcomes.

2.   Does system feedback support the execution of the subgoal?

The system should provide feedback that the chosen action or command is acting on the object of interest. Also, the system should provide feedback to indicate clearly whether a subgoal has been executed successfully.

3.   Does the interface provide error recovery procedures?

The interface should provide obvious and readily available error recovery procedures so that users can reverse the system state to an easily recognizable previous state.

Step 6. Final Evaluation

The designer should put the revised interface through its paces by asking users in the target domain to use the system. Their feedback and performance bottlenecks should further help refine the final design. They should be asked whether the interface allows them to decompose and execute the tasks that feels 'right' and 'natural' to them. Their keystroke time can be checked for unusual long pauses or pauses at the wrong locations against the keystroke level model's parameters.

APPENDICES

APPENDIX A

THE INSTRUCTIONS FOR THE FORMULA EDITOR PILOT STUDY

Instructions for Subjects

Your task is very simple. It is just to key in 20 formulas. The only difficult part might be having you insert parentheses so that the operations are done in the right order as conveyed in the formula.

You only need to use a restricted set of keys, mainly the keys on the numeric keypad and a few other keys nearby. The variables are only the numbers 1, 2, 3, 4, 5, 6. All the keystrokes are simple keystrokes. <shift> and <ctrl> keys are not needed. Please take a few minutes to familiarize yourself with the keyboard. Go through all the relevant keys with white key labels on them.

< <-- >	deletes the previous character at the cursor position.
<delete>	deletes the character at the cursor.
<eol>	brings the cursor to the end of the line.
<start>	clears the window before keying in a new formula.
<0.5>	inserts 0.5 for square root operation.

Inserting a character within a string takes effect at the cursor position and will push all characters a position to the right starting at the cursor position.

The formulas are presented to you one at a time on a card (show card). At the beginning of the trial, press <start>, then look at a card. When you are ready just begin typing. When you are done, just hit the <return> key. Press <start> again and look at the next card, and so on.

The experiment has three parts. Please inform the experimenter when you are done with each part or when you encounter any difficulties.

1.   Practice Run

2.   Actual Trials

3.   Circle conceptual chunks on formula sheets.

Please take some time to look at each formula and cricle chunks within the formula you feel are the things that go together that form conceptually complete units. You can make circles that overlaps or within other circles.

APPENDIX B

FORMULAS IN THE FORMULA EDITOR PILOT STUDY

I.   PRACTICE TRIALS

$1+2*3/4+5^2$

$(1*2)^3$

$1+2*3/4+\sqrt{3+6}$

$(1+2)*(3+4)$

$1*[(2+3)*(5-6)]$

$\dfrac{3+4}{5+6}$

$\dfrac{\sqrt{\left(\dfrac{3+4}{5}\right)}}{2+3}$

$\dfrac{(3+5)*6}{(3+2)*(6+5)}$

$\left(\dfrac{3+4}{5}\right)+6*\left(\dfrac{2+3}{4}\right)+[3*(7+2)]^2$

$1+\{(2*3)+[2*(3+[4*\dfrac{5}{4+2}])]\}$

II. ACTUAL TRIALS

1. $$\frac{\sqrt{1+2}*\left\{\frac{1}{2+3}\right\}}{4+(5*6)^2}$$

2. $$\frac{\left\{[(3+5)*6]/5\right\}^2}{(3*2)+(6*5)}$$

3. $$\frac{2+3*\left\{\left(\frac{2+3}{4}\right)^5+2\right\}}{2}$$

4. $$\frac{5*\left(\frac{3}{1+2}\right)+\frac{5}{6}}{\sqrt{\frac{4}{2+3}}}$$

5. $$\frac{2*\left\{\sqrt{1+2}*\left(\frac{3}{4+5}\right)\right\}}{1+(2*3)^2}$$

6. $$\frac{3+(2*3)^{4}}{\sqrt{2+4}*\left\{\dfrac{2}{4+2}\right\}}$$

7. $$\frac{\dfrac{(2*4)+(3*2)}{4}}{\{[(2+2)*3]/2\}}$$

8. $$\frac{4}{4+2*\left\{\left(\dfrac{4+2}{3}\right)^{2}+4\right\}}$$

9. $$\frac{\sqrt{\dfrac{3}{4+2}}}{2*\left(\dfrac{2}{2+4}\right)+\dfrac{2}{3}}$$

10. $$\frac{2+(4*2)^{4}}{4*\left\{\sqrt{2+4}*\left(\dfrac{2}{3+2}\right)\right\}}$$

APPENDIX C

AN EXAMPLE OF THE FORMULA CHUNKING RESULTS

Please cirlce the conceptual chunks

$$(5*(3/(1+2)+5/6))/((4/(2+3))^.5)$$

$$3+\left\{\sqrt{\frac{1+2}{4}*6/7}+3\right\}*8$$

$$(5*((4+2)/(5+2))*3)^.5/(3*(4+5)/2)$$

$$\sqrt{\frac{5*\frac{4+2}{5+2}*3}{3*\frac{4+5}{2}}}$$

$$(2+(4*2)^4)/(4*((2+4)^.5*(2/(3+2))))$$

APPENDIX D

THE FORMULAS IN THE FORMULA EDITOR EXPERIMENT

1. $$\dfrac{6\text{-}4}{2}$$

2. $$\dfrac{1}{4\text{+}3}$$

3. $$\dfrac{\left(\dfrac{3\text{+}5}{3}\right)^{2}\text{*}2}{(1\text{+}2)\text{*}4}$$

4. $$\dfrac{\sqrt{1\text{+}2}\text{*}\left\{4\text{+}\dfrac{1}{2\text{+}3}\right\}}{4\text{*}(5\text{+}6)}$$

5. $$\dfrac{(6\text{+}2)\text{*}3}{\sqrt{4\text{+}5}\,\text{*}4}$$

6. $$\dfrac{\dfrac{4\text{+}2}{2}}{(1\text{+}5)\,\text{*}\,(1\text{+}3)}$$

7. $$\left[(3\text{+}1)\,\text{*}\,\dfrac{4}{2\text{+}3}\right]^{3}$$

8. $$\dfrac{(1+2)\ *\ 5}{\sqrt{\dfrac{4+3}{2+3}}}$$

9. $$\dfrac{[(3+5)*6]^{5}\ +4}{(3+2)*(6+5)}$$

10. $$\dfrac{2}{4*\left(3+\dfrac{5}{4*(4-2)}\right)}$$

11. $$\left[\dfrac{2+4}{4*\left\{2+\dfrac{2}{3+2}\right\}}\right]^{3}$$

12. $$\dfrac{(2+4)*3}{\{\sqrt{(2+2)*3}\ +5\}*2}$$

13. $$\dfrac{(2*3)^{4}\ +3}{2*4+\left\{\dfrac{4}{2+3}\right\}}$$

14. $$\dfrac{(3+5)\ *\ 3}{5*\dfrac{3-1}{4+6}}$$

15. $$\dfrac{4}{2*\left\{\left(\dfrac{4+2}{3}\right)^{2}+4\right\}}$$

16.
$$\frac{\sqrt{4+2}}{2 * \left(2 + \dfrac{2}{2+4}\right)}$$

17.
$$3 * \left\{ \sqrt{\left(\dfrac{1+2}{4}\right) * 6} + \dfrac{3}{5+2} \right\}$$

18.
$$\frac{(6-2) * \left\{ \sqrt{1 * 2} \middle/ \dfrac{3}{4+5} \right\}}{3}$$

19.
$$\frac{\left\{ \left(\dfrac{2+3}{3} + 2\right) * \left(3 + 3\right) \right\}^{2} + 3}{1+2}$$

20
$$\sqrt{\dfrac{1}{5 * \left\{ \dfrac{1}{5 + \dfrac{2}{5-3}} \right\}}}$$

APPENDIX E

THE CLASSIFICATION OF FORMULAS USED IN THE
EDITOR EXPERIMENT

<u>Independent Variable</u>:   Depth (2,3,4,5), Skewness (left, right), Decay (short, long)

(Note: The number following the classification below, e.g., Left Short: 5, is the Formula number in the experiment. See Appendix D for a listing of the typeset formulas present in the experiment.)

Depth: 2

Left Short:5	$((6+2)*3)/\underline{((4+5)^2*4)}$
Left Long:6	$(4+2)/\underline{((1+5)*(1+3)^2)}$
Right Short:13	$((2+4)*3)/\underline{(2*4+4/(2+3))}$
Right Long:7	$\underline{((3+1)*4/(2+3))}^3$

Depth: 3

Left Short:9	$\underline{(((3+5)*6)^5+4)}/((3+2)*6)$
Left Long:8	$((1+2)*5)/\underline{(((4+3)/(2+3))^2)}$
Right Short:16	$((4+2)^2)/\underline{(2*(2+2/(2+4)))}$
Right Long:14	$((3+5)*3)/\underline{(5*((3-1)/(4+6)))}$

Depth: 4

Left Short:15   4 / (2+ ((((4+2) / 3) ^2+4))

Left Long:17   3 * ((((1+2) / 4 * 6) ^2) +3 / (5+2))

Right Short:11   ((2+4) / (4 * (2+2 / (3+2)))) ^3

Right Long:18   ((1-2) * ((1+2) ^2) / (3 / (4+5))) / 3

Depth: 5

Left Short:12   (2+4) * 3 / (((((2+2) * 3) ^2) +5) * 2)

Left Long:19   (((((2+3) / 3+2) * (3+3)) ^2) +3) / (1+2)

Right Short:10   2 / (4 * (3+ (5 / (4 * (4-2)))))

Right Long:20   (1 / (5 * (1 / (5+2 / (5-3))))) ^2

APPENDIX F

THE INSTRUCTIONS FOR THE PRACTICE SESSION IN
THE EDITOR EXPERIMENT

<u>GENERAL INSTRUCTIONS</u>

Explain formulas, its semantic units. (Show Grammar slide)

(Show Formula-slide). Your task is just to key in formulas like this.

Do not evaluate the formula, eg., do not type in 2+3 as 5. Do not rearrange the formula, eg. $\frac{1}{2*3}$ is 1/(2*3) and not 1/2/3.

Enclose each complex object with parentheses.

$5*\frac{1+2}{2+3}$ is 5*((1+2)/(2+3))

Your task is just to type in some formulas using two different editors. One editor will be one you are very familiar with, as used in spreadsheet programs, a left-to-right **linear** editor. The other editor, which works with the semantic units of the formula, is a **semantic** editor.

You will use only the keys with stickers on them. The keys are color coded: operators, variables, cursor movement keys, parentheses, undo and start and finish keys. The variables are only the numbers 1, 2, 3, 4, 5, 6. ^ means raise to the power of, eg. $\sqrt{1+2}$ is (1+2)^.5. The <BkSp> deletes the character immediately to the left of the cursor.

All the keystrokes are simple keystrokes. <shift> and <ctrl> keys are not needed. (Show Keyboard Map).

The formulas are presented to you one at a time (show card) except during the training session. Before looking at the formula, press <start>. Look carefully at the formula and make sure you understand its structure before you start typing. When you are done with the formula, press the <end> key. Press <start> again and look at the next formula and continue likewise till the last formula. Do not spend time checking the correctness of the formula once you are done. Work as quickly and as accurately as you can.

(Demonstrate Linear editor)

(Demonstrate Semantic Editor)

Instructions for Linear Editor

You are going to use an ordinary text editor to type in the formulas in a left to right fashion. The left and right arrows keys and the eol, sol keys are provided for you to move around in the formula.

Let's go through some examples. (Let subjects go through Warm Up session A & B)

The purpose of the experiment is not to test your knowledge of the precedence of operators. Type the parentheses as you see them in the formula on the card. Insert extra parentheses around complex operants which do not have parentheses around them.

Go through the rest of the practice formulas.

Go through dry run.

Instructions for Semantic Editor

You are going to learn how to use an editor that works with the operants of the formula as units. This will eliminate the need to type the parantheses explicitly.

Parantheses in formulas are used to delimit complex operants. Formulas are built up recursively with the fundamental structure of <operant><operator><operant>.

(Go through example on overhead and work out example by hand)

The editor utilizes the concept by allowing you to work with operants as units in a left to right fashion. A special key <()> is used to create complex operant>.

(Demonstration of how complex operants are created with <()>, relate them back to concept on paper)

(Let subjects go through training session A, B, C and D)

Notice that you never need to close unbalanced parentheses and you never need to plan for open parentheses needed. The only preplaning is to use <()> after an operator whenever a complex operant is to follow the operator.

(Show undo capability of editor)

(Go through session E)

(Go through session F)

I. Warming Up for Linear Editor

(Fonts and spacings between formulas are much bigger in actual material used by subjects)

A. Familiarization with keyboard

1+2+3+4+5+6

$1*2-3+4/5+\sqrt{6}$

B. Typing parentheses as they are presented

(1+2)*3

[(1+2)*3]^2

1*{ 2+[3/(4-5)] }

C. More practice with some needs to create extra parantheses

$\dfrac{3-2}{5+2}$ *4

Ans: ((3-2)/(5+2))*4

$\dfrac{3+4}{(5+6)*(1+2)}$

Ans: (3+4)/((5+6)*(1+2))

$$\frac{\sqrt{\frac{3+4}{6}}}{2+3}$$

Ans: (((3+4)/6)^.5)/(2+3)

D.   Complex Formula

$$1+\left\{\frac{2+3}{2+\sqrt{\frac{3}{4+2}}}\right\}$$

Ans: 1+((2+3)/(2+((3/(4+2))^.5)))

$$\frac{\left(\frac{1}{4\text{-}2}+5\right)*(6\text{-}2)^2}{[\,(5*1)^2+6]\,*4}$$

Ans: (((1/(4-2))+5)*((6-2)^2))/((((5*1)^2)+6)*4)

$$\frac{\left[3*\left\{\left(\frac{2+3}{4}\right)^5+2\right\}\right]^2}{2}$$

Ans: ((3*((((2+3)/4)^5)+2))^2)/2

$$\frac{\left(\frac{5\text{-}1}{3}+\frac{1}{2\text{-}5}\right)*2}{\sqrt{\frac{3\text{-}2}{3}}}$$

Ans: ((((5-1)/3)+(1/(2-5)))*2)/(((3-2)/3)^.5)

I.   Warming Up for Semantic Editor

(Fonts and spacings between formulas are much bigger in actual material used by subjects)

A.   Familiarization with keyboard

1+2+3+4+5+6

$1*2-3+4/5+\sqrt{6}$ same as in the linear editor

B.   Creating Left Complex Operand (ie left parenthesis) by using '()' key before operator

(1+2)*3

key: 1+2 P*3

[(1+2)*3]^2

key: 1+2 P*3 P^2

[(1+2)*3+4+5]*6   Note: Consecutive highlighting to create operant

Key: 1+2 P*3 P+4 +5 PP*6

Note: you never have to plan for left parentheses, they are created automatically when complex operant is created.

C.   Creating right complex operant by using '()' key after operator

$1*(2+3)$

key: 1*P 2+3

$1*\{ 2+[3/(4-5)] \}$

key: 1*P 2+P 3/P 4-5

          The parentheses are automatically created when the complex
operant is created.

D.   Left and right complex operants

$(1+2)*(3+4)$

key: 1+2 P*P 3+4

$1*[(2+3)*(5-6)]$ Note: 2+3 is keyed in before making it (2+3)*

                            Never create two complex operants in a row

Key: 1*P 2+3 P*P 5-6

$1*(\{ [(2+3)*4]+5 \}*6)$

key: 1*P 2+3 P*4 P+5 P*6

$$\frac{3+4}{(5+6)*(1+2)}$$

Note: Operant before operatore created last,
operant after operator created first

Key: 3+4 **P/P** 5+6 P*P 1+2

$$\frac{\sqrt{\dfrac{3+4}{6}}}{2+3}$$

Key: 3+4 P/6 P^.5 P/P 2+3

$$\frac{\sqrt{\dfrac{3+4}{6}*(1+2)}}{2+3}$$

Key: 3+4 P/6 P^.5 P*P 1+2 **PPP/P** 2+3

E.   Correcting Mistakes

Undo Facility

1.   Undo highlighting

     1+2+3+4

2.   Undo Operator and parentheses created after highlighting and pressing the operator key.

3.   Undo parenthese created after operator.

     1+(_)

4.   Simply backspace for wrong variable or operator.

5.   No backspace over parentheses, need to retype

F.   Complex Formula

$$1+\left\{\cfrac{2+3}{2+\sqrt{\cfrac{3}{4+2}}}\right\}$$

Ans: 1+((2+3)/(2+((3/(4+2))^.5)))

Key: 1+P 2+3 P/P 2+P 3/P 4+2 P^.5

$$\cfrac{\left(\cfrac{1}{4-2}+5\right)*(6-2)^{2}}{[\,(5*1)^{2}+6]\,*4}$$

Ans: (((1/(4-2))+5)*((6-2)^2))/((((5*1)^2)+6)*4)

Key: 1/P 4-2 PPP+5 P*P 6-2 P^2 PPPP/P 5*1 P^2 P+6 P*4

$$\cfrac{\left[3*\left\{\left(\cfrac{2+3}{4}\right)^{5}+2\right\}\right]^{2}}{2}$$

Ans: ((3*((((2+3)/4)^5)+2))^2)/2

Key: 3*P 2+3 P/4 P^5 P+2 PPP^2 P/2

$$\cfrac{\left(\cfrac{5-1}{3}+\cfrac{1}{2-5}\right)*2}{\sqrt{\cfrac{3-2}{3}}}$$

Ans: ((((5-1)/3)+(1/(2-5)))*2)/(((3-2)/3)^.5)

Key: 5-1 P/3 P+P 1/P 2-5 PPPPP*2 P/P 3-2 P/3 P^.5

APPENDIX G

THE TASKS AND SPREADSHEETS IN THE LOTUS STUDY

Tasks on Acma

1   Change the width of column A to 5

        Change the width of column B to 26

2.   Change the default column width to 4

3   Delete the row labeled "other Sources"

4   Copy the formula for "Ending Cash Balance" in cell c27 to cell d26

5   Insert a column before column a

6   Make Default label justification: Left

7   Center all the column Headings (Ds..E4)

8   Make default Formats for numbers: Fixed with 0 decimal point

9   Format the two numbers in "Finding Cash Balance" as currency with 2 decimal point

10   Erase the two numbers in Dividends

ACMA COMPANY
CASH BUDGET

	(November) 1984	(December) 1984
Cash Receipts		
Cash Sales	$2,085.00	$3,764.00
Collections from Customers	$4,661.00	$1,383.00
Short-term Borrowing	$2,115.00	$4,685.00
Other Sources	$3,836.00	$14,677.00
Other Current Assets	$6,484.00	$8,666.00
Total Cash Receipts	$19,181.00	$33,175.00
Cash Disbursements		
Manufacturing Costs	$9,225.00	$8,353.00
Operating Expenses	$2,699.00	$10,937.00
Capital Expenditures	$4,569.00	$13,034.00
Dividends	$12,914.00	$8,311.00
Other Disbursements	$7,324.00	$14,745.00
Total Cash Disbursements	$36,731.00	$55,380.00
Net Cash Provided (applied)	($17,550.00)	($22,205.00)
Beginning Cash Balance	$20,000.00	$2,450.00
ENDING CASH BALANCE	2450.00	

Tasks on Best

1   Insert a row before "year Ending"

2   Right Justify the column heading in (D3..E4)

3   Move Column Headings (A5..E6) to the bottom of the spreadsheet

4   Make Default label Justificaiton: Center

5   Change the width of Column C to 32

    Change the width of Column D,E,F to 10

6   Change the Dafault column width to 8

7   Erase the single line under Revenue

8   Delete Column A

9   Make Default formats for numbers: Currency with 0 decimal point

10   Format the numbers in "Revenue" as Currency with 2 decimal points

```
BEST COMPANY
INCOME STATEMENT
```

Year Ending	April 1984	April 1985	April 1986
Revenue	8300	9876	8765
Costs of Goods Sold			
Materials	562	713	71
Salaries	43	404	831
Fringe Benefits	653	368	430
Others	281	726	419
General and Adminstrative Expense			
Compensation: Offics	463	984	495
Compensation: Sales	575	665	710
Fringe Benefits	580	57	759
Advertising and Promotio	794	199	362
Depreciation	91	37	49
Miscellaneous	53	740	162
Total Operating Expenses	4095	4893	4288
NET INCOME	4205	4983	4477

Tasks on Cost

1   Delete Row 6, an empty Row.

2   Make Default Format for numbers: currency with 0 decimal point

3   Copy Column Headings (C3..E4) to bottom of spreadsheet

4   Format the numbers in Column E as percentage with one decimal point

5   Change the default column width to 15

6   Change the width of column C&D to 12

7   Insert a row Before "Total Assets" (Row 23)

8   Erase the numbers in the row "Investments" (Row 19)

9   Center the Column Headings (C3..E4)

10   Make Default Label Justification: Center

COST SAVER SUPERMARKET
COMPARATIVE BALANCE SHEET

	Year of 1989	Year of 1990	% Increase or Decrease
Current Assets:			
Cash	9530.0	8636.0	-0.09
Marketable Securities	10163.0	6902.0	-0.32
Accounts Receivable: Trade	7648.0	19385.0	1.53
Accounts Receivable: Other	7538.0	10412.0	0.38
Inventories	6122.0	1623.0	-0.73
Prepaid Expenses	13337.0	1713.0	-0.87
Other Current Assets	4050.0	14233.0	2.52
Total Current Assets	58388.0	62959.0	0.08
Long-term Assets:			
Property, Plant & Equiment	17753.0	12983.0	-0.27
Investments	4089.0	1099.0	-0.73
Total Long-term Assets	21842.0	14082.0	-0.36
TOTAL ASSETS	80230.0	77041.0	-0.04

BIBLIOGRAPHY

183

BIBLIOGRAPHY

Allen, R.B. and Scerbo, M.W. (1983). "Details of command-language keystrokes." *ACM Transactions on Office Information Systems* 1: 159-178.

Allport, D.A. (1980). Patterns and actions: cognitive mechanisms are content specific. Cognitive psychology: new directions. London, Rouledge & Kegan Paul. 26-64.

Anderson, J.R. (1983). The architecture of cognition. Mass., Harvard University Press.

Anderson, J.R. and Jeffries, R. (1985). "Novice LISP errors: undetected losses of information from working memory." *Human-computer Interaction* 1(1): 107-131.

Barnard, P.,Wilson, M., & Maclean, A. (1987). Approximate modelling of cognitive activity: towards an expert system design aid. In J. M. Carroll & P. P. Tanner (Ed.), CHI'87 Human Factors in Computing Systems, New York: ACM Press.

Barnard, P.J. (1987). Cognitive resources and the learning of human-computer dialogs. Interfacing thought. Cambridge, Bradford Book, MIT Press.

Bellotti, V. (1987). Implications of Current Design Practice for the Use of HCI Techniques. People and computers IV. Proceedings of the Fourth Conference of the BCS HCI Specialist Group. University of Manchester, 5-9 September 1988. Cambridge, Cambridge University Press. 46-61.

Bertino, E. (1985). "Design issues in interactive user interfaces." *Interfaces in Computing* 3: 37-53.

Black, J.B., Kay, D.S., et al. (1987). Goal and plan knowledge representations: from stories to text editors and programs. Interfacing thought. Cambridge, Bradford Book, MIT Press.

Booth, P. (1989). An introduction to human-computer interaction. Hillsdale, Lawrence Erlbaum Associates.

Bower, G.H., Clark, M.C., et al. (1969). "Hierarchical retrieval schemes in recall of categorized word lists." *Journal of Verbal Learning and Verbal Behavior* 9: 323-343.

Brown, P.S. and Gould, J.D. (1987). "An experimental study of people creating spreadsheets." *ACM Transactions on Office Information Systems* 5(3): 258-272.

Campbell, R.L. (1990). Developmental scenario analysis of smalltalk programming. CHI'90 Human Factors in Computing Systems, Seattle, Washigton, ACM Press.

Card, S.K., Moran, T.P., et al. (1980). "Computer text-editing: an information-processing analysis of a routine cognitive skill." *Cognitive Psychology* 12: 32-74.

Card, S.K., Moran, T.P., et al. (1980). "The Keystroke-level model for user performance time with interactive systems." *Communications of the ACM* 23: 396-410.

Card, S.K., Moran, T.P., et al. (1983). The psychology of human-computer interaction. Hillsdale, NJ, Lawrence Erlbaum Associates, Inc.

Carroll, J.M. (1990). Infinite detail and emulation in an ontologically minimized HCI. CHI'90 Human Factors in Computing Systems, Seattle, Washigton, ACM Press.

Carroll, J.M. and Campbell, R.L. (1986). "Softening up hard science: reply to Newell and Card." *Human-computer Interaction* 2: 227-250.

Cornsweet, T.N. (1970). Visual Perception. New York, Academic Press.

Cypher, A. (1986). The structure of users' activities. User centered system design: new perspectives on human-computer interaction. Hillsdale, NJ, Erlbaum Associates.

Dertouzos, M.L. (1990). Redefining tomorrow's user interface. CHI'90 Human Factors in Computing Systems, Seattle, Washigton, ACM Press.

Ditlea, S. (1987). Spreadsheets can be hazardous to your health. *Personal Computing*. 62-67.

Elkerton, J., & Palmiter, S. L. (1991). Designing help using a GOMS model: an information retreival evaluation. *Human Factors*, 33(2), 185-204.

Filkes, R.E. (1982). "A commitment-based framework for describing co-operative work." *Cognitive science* 6: 331-347.

Fitts, P.M. (1964). Perceptual-motor skill learning. Categories of human learning. New York, Academic Press.

Fitts, P.M. and Posner, M.L. (1967). Human Performance. Belmont, CA, Brooks/Cole Publishing.

Fleishman, E.A. and Quaintance, M.K. (1984). Taxonomies of Human Performance. Orlando, Florida, Academic Press.

Fountain, A.J. and Norman, M.A. (1985). Modelling user behavior with formal grammar. People and computers: Designing the interface. Proceedings of the Conference of the BCS HCI Specialist Group. University of East Anglia, 17-20 September 1985. Cambridge, Cambridge University Press. 3-12.

Gentner, D.R. and Grudin, J. (1990). Why good engineers (sometimes) create bad interfaces. CHI'90 Human Factors in Computing Systems, Seattle, Washigton, ACM Press.

Goel, V., & Pirolli, P. (1989). Motivating the notion of generic design within information-processing theory: the design problem space. *AI Megazine, 2*, 18-36.

Goodstein, L., Andersen, H., et al. (1986). Introduction. Tasks, Errors, and Mental Models. London, Taylor & Francis.

Grudin, J. (1990). The computer reaches out: The historical continuity of interface design. CHI'90 Human Factors in Computing Systems, Seattle, Washigton, ACM Press.

Holland, J.H. (1986). "A mathematical framework for studying learning in classifier systems." *Physica* 22D: 307-317.

Hoppe, H.U. (1988). Task-oriented parsing - a diagnostic method to be used by adaptive systems. CHI'88 Human Factors in Computing Systems, Washigton D. C., ACM Press.

Hutchins, E.L., Hollan, J.D., et al. (1986). Direct manipulation interfaces. User Centered System Design: New Perspectives on Human-Computer Interaction. Hillsdale, NJ, Lawrence Erlbaum Associates, Inc.

John, B.E. (1988). Contributions to engineering models of human-computer interaction. Phd thesis, Carnegie Mellon University.

John, B.E. (1990). Extensions of GOMS analyses to expert performance requiring perception of dynamic visual and auditory information. CHI'90 Human Factors in Computing Systems, Seattle, Washigton, ACM Press.

Johnson, P. (1985). Toward a task model of messaging: an example of the application of TAKD to user interface design. People and computers: Designing the interface. Proceedings of the Conference of the BCS HCI Specialist Group. University of East Anglia, 17-20 September 1985. Cambridge, Cambridge University Press. 46-61.

Johnson, P., Johnson, H., et al. (1988). Task-related knowledge structures: analysis, modelling and application. People and computers IV. Proceedings of the Fourth Conference of the BCS HCI Specialist Group. University of Manchester, 5-9 September 1988. Cambridge, Cambridge University Press. 46-61.

Kasprzyk, D.M., Drury, C.G., et al. (1979). "Human Behaviour and performance in calculator use with Algebraic and Reverse Polish Notation." *Ergonomics* 22(9): 1011-1019.

Kelley, H.H. (1973). "The processes of causal attribution." *American Psychology* 28(2): 107-128.

Kieras, D.E. (1988). Towards a practical GOMS model methodology for user interface design. Handbook of Human-Computer Interaction. North-Holland, Elsevier Science Publishers B. V. 905-928.

Kieras, D.E. and Bovair, S. (1986). "The acquisition of procedures from text: a production-system analysis of transfer of training." *Journal of Memory and Learning* 25: 507-524.

Kieras, D.E. and Polson, P. (1985). "An approach to the formal analysis of user complexity." *International Journal of Man-Machine Studies* 22(4): 365-394.

Knowles, C. (1987). Can Cognitive Complexity Theory (CCT) produce an adequate measure of system usability. People and computers IV. Proceedings of the Fourth Conference of the BCS HCI Specialist Group. University of Manchester, 5-9 September 1988. Cambridge, Cambridge University Press. 46-61.

Laird, J. E.,Newell, A., & Rosenbloom, P. S. (1987). Soar: An Architecture for general intelligence. *Artificial Intelligence*, 33, 1-64.

Landauer, T.K. (1987). Relations between cognitive psychology and computer system design. Interfacing thought. Cambridge, Bradford Book, MIT Press.

Landauer, T.K. (1988). Research methods in human-computer interaction. Handbook of Human-Computer Interaction. North-Holland, Elsevier Science Publishers B. V. 905-928.

Lerch, F.J. (1988). Computerized Financial Planning: discovering cognitive difficulties in model building. PhD thesis, University of Michigan.

Lewis, C.H. (1988). "How and why to learn why: analysis-based generalization of procedures." *Cognitive Science* 12: 211-256.

Lewis, C.H. (1990). "A research agenda for the nineties in human-computer interaction." *Human-Computer Interaction* 5: 125-143.

Liebelt, L.S., McDonald, J.E., et al. (1982). The effect of organization on learning menu access. Human factors society - 26th annual Meeting, 1982,

Lohse, J. (1991). A Cognitive Model for the Perception and Understanding of Graphs. In S. P. Robertson,G. M. Olson, & J. S. Olson (Ed.), CHI'90 Human Factors in Computing Systems, New Orleans, Louisiana: ACM Press.

Mayes, T.J., Draper, S.W., et al. (1987). Information flow in a user interface: the effect of experience and context on the recall of MacWrite screens. People and computers IV. Proceedings of the Fourth Conference of the BCS HCI Specialist Group. University of Manchester, 5-9 September 1988. Cambridge, Cambridge University Press. 46-61.

Meister, D. (1976). Behavioral foundations of system development. New York, John Wiley & Sons.

Miller, L.A. (1956). "The magical number seven plus or minus two: Some limits on our capacity for processing information." *Psychological Review* 63: 81-97.

Moran, T.P. (1981). "The Command Language Grammar: a representation for the user interface of interactive computer systems." *International Journal of Man-Machine Studies* 15: 3-50.

Moran, T.P. (1983). Getting into a system: External-internal task mapping analysis. CHI'83 Human Factors in Computing Systems, Boston, North-Holland.

Moran, T.P. (1986). Analytical performance models: a contribution to a panel discussion. Human factors in computer systems-III, Boston, Elsevier.

Neves, D.M. and Anderson, J.R. (1981). Knowledge compilation: Mechanisms for the automation of cognitive skills. Cognitive Skills and Their Acquisition. Hillsdale, NJ, Lawrence Erlbaum.

Newell, A. and Simon, H.A. (1972). Human Problem Solving. Englewood Cliffs, N.J., Prentice-Hall.

Newell, A., & Rosenbloom, P. S. (1981). Mechanisms of skill acquisition and the law of practice. In J. R. Anderson (Eds.), Cognitive Skills and their Acquisition Hillsdale, New Jersey: Lawrence Erlbaum Associates.

Nilsen, E. L. (1991). Perceptual-motor control in human-computer interaction. PhD thesis, The University of Michigan.

Norman, D.A. (1986). Cognitive engineering. User centered system design: new perspectives on human-computer interaction. Hillsdale, NJ, Erlbaum Associates.

Norman, D.A. (1987). Cognitive engineering, cognitive science. Interfacing thought. Cambridge, Bradford Book, MIT Press.

Olson, J.R. and Nilsen, E. (1988). "Analysis of the cognition involved in spreadsheet software interaction." *Human Computer Interaction* 3(4): 309-350.

Olson, J.R. and Olson, G.M. (1990). "The growth of cognitive modeling in human-computer interaction since GOMS." *Human Computer Interaction* 5: 221-265.

Payne, S.J. and Green, T.R.G. (1986). "Task action grammars: a model of the mental representation of task languages." *Human Computer Interaction* 2: 93-133.

Polson, P. (1987). A quantitative theory of human-computer interaction. Interfacing thought. Cambridge, Bradford Book, MIT Press.

Polson, P.G. and Lewis, C.H. (1990). "Theory-based design for easily learned interfaces." *Human Computer Interaction* 5: 191-220.

Rasmussen, J. (1976). Outlines of a hybrid model of the process operator. Monitoring Behavior and Supervisory Control. New York, Plenum Press.

Rasmussen, J. (1980). The human as a system component. Human interaction with Computers. London, Academic Press.

Rasmussen, J. (1986). On information processing and Human-Machine Interaction: an approach to cognitive engineering. Amsterdam, Elsevier.

Reisner, P. (1987). Discussion: HCI, what is it and what research is needed? Interfacing thought. Cambridge, Bradford Book, MIT Press.

Reisner, R. (1981). "Formal grammar and human factors design of an interactive graphics system." IEEE Transaction on Software Engineering SE-7(2): 229-140.

Reisner, R. (1984). Formal grammar as a tool for analyzing ease of use: some fundamental concepts. Human Factors in Computer Systems. New Jersey, Ablex Pub. Corp.

Reitman, W. R. (1965). Cognition and thought. New York: Wiley.

Roberts, T.L. and Moran, T.P. (1982). Evaluation of text editors. Human Factors in Compute Systems, Gaithersburg, ACM Press.

Sanderson, P.M. and Harwood, K. (1986). The skills, rules and knowledge classification: a discussion of its emergence and nature. Tasks, errors, and mental models. London, Taylor & Francis.

Schank, R.S. and Abelson, R.P. (1977). Scripts, Plans, Goals, and Understanding. Hillsdale, N.J., Lawrence Drlbaum & Assoc.

Schmidt, R.A. (1982). Motor control and learning — a behavior emphasis. champaign, Illinois, Human Kinetics Publishers.

Sekular, R. and Blake, R. (1985). Perception. New York, Knopf.

Shackel, B. (1985). "Ergonomics in information technology in Europe - a review." Behaviour and Information Technology 4(4): 263-287.

Shapiro, A.D. (1987). Structured induction in expert systems. New York, Addison-Wesley.

Shiffrin, R.M. and Dumais, S.T. (1981). The development of automatism. Cognitive Skills and Their Acquisition. Hillsdale, New Jersey, Lawrence Erlbaum Associates, Publishers.

Simon, T. (1987). Analyzing the scope of cognitive models in human-computer interaction: a trade-off approach. People and computers IV. Proceedings of the Fourth Conference of the BCS HCI Specialist Group. University of Manchester, 5-9 September 1988. Cambridge, Cambridge University Press. 46-61.

Simon, T. and Young, R.M. (1988). GOMS meets STRIPS: the integration of planning with skilled procedure execution in human-computer interaction. People and computers IV. Proceedings of the Fourth Conference of the BCS HCI Specialist Group. University of Manchester, 5-9 September 1988. Cambridge, Cambridge University Press. 46-61.

Slobin, D.I. (1979). Psycholinguistics. Glenview, Illinois, Scott, Foresman and Company.

Smelcer, J.B. (1989). Understanding user errors in database query. PhD thesis, University of Michigan.

Straub, D.W. and Wetherbe, J.C. (1989). "Information technologies for the 1990s: an organizational impact perspective." *Communications of the ACM* 32(11): 1328-1339.

Sutcliffe, A. (1987). Some experience in integrating specification of human computer interaction within a structured system development. People and computers IV. Proceedings of the Fourth Conference of the BCS HCI Specialist Group. University of Manchester, 5-9 September 1988. Cambridge, Cambridge University Press. 46-61.

Swets, J.A. (1964). Signal detection and recognition by human observers. New York, Wiley.

Tversky, A. and Kahneman, D. (1974). "Judgement under uncertainty: Heuristics and biases." *Science* : 1123-1124.

Waem, Y. (1989). Cognitive aspects of computer supported tasks. Essex, John Wiley & Sons.

Waugh, N.C. and Norman, D.A. (1965). "Primary memory." *Psychological Review* 72: 89-104.

Wilson, M.D., Barnard, P.J., et al. (1985). Analysing the learning of command sequences in a menu system. People and computers: Designing the interface. Proceedings of the Conference of the BCS HCI Specialist Group. University of East Anglia, 17-20 September 1985. Cambridge, Cambridge University Press. 63-75.

Wilson, M.D., Barnard, P.J., et al. (1988). Knowledge-based analysis for human-computer systems. Working with Computers: Theory versus Outcome. London, Academic Press. Gaines, B.R., ed. 47-87.

Young, M.R. and Simon, P. (1987). Planning in the context of human-computer interaction. People and computers IV. Proceedings of the Fourth Conference of the BCS HCI Specialist Group. University of Manchester, 5-9 September 1988. Cambridge, Cambridge University Press. 363-370.

Young, R.M. (1981). "The machine inside the machine, users' models of pocket calculators." *International Journal of Man-Machine Studies* 15: 51-85.

Young, R.M. and Whittington, J. (1990). Using a knowledge analysis to predict conceptual errors in text-editor usage. CHI'90 Human Factors in Computing Systems, Seattle, Washigton, ACM Press.

Cognitive Science and Machine Intelligence Laboratory
The University of Michigan
Ann Arbor, Michigan

Technical Report Series

1. Combining Prototypes: A Modification Model. Edward E. Smith, Daniel N. Osherson, Lance J. Rips and Margaret Keane. January 1987.

2. Soar: An Architecture for General Intelligence. John E. Laird, Allen Newell and Paul S. Rosenbloom. January 1987.

3. Some Origins of Belief. Daniel. N. Osherson, Edward E. Smith and Eldar B. Shafir. January 1987.

4. Adaptive Information Retrieval: Machine Learning in Associative Networks. Richard K. Belew. January 1987.

5. Activation and Metacognition of Inaccessible Stored Information: Potential Bases for Incubation Effects in Problem Solving. Ilan Yaniv and David E. Meyer. January 1987.

6. Structure and Process in Semantic Memory: New Evidence Based on Speed Accuracy Decomposition. John Kounios, Allen Osman, and David E. Meyer. April 1987

7. New Concepts in Tele-Autonomous Systems and Tele-Autonomous Systems: Methods and Architectures for Intermingling Autonomous and Telerobotic Technology. Lynn Conway, Richard Volz and Michael Walker. April 1987.

8. Classifier Systems and Genetic Algorithms. L.B. Booker, D.E. Goldberg and J.H. Holland. April 1987.

9. LFP: A Logic for Linguistic Descriptions and An Analysis of its Complexity. William C. Rounds. May 1987.

10. Fluid Concepts and Creative Analogies: A Theory and Its Computer Implementation. Douglas R. Hofstadter, Melanie Mitchell, and Robert M. French. May 1987.

11. A Qualitative Approach for Recovering Relative Depths in Dynamic Scenes. Susan M. Haynes and Ramesh Jain. June 1987

12. Mental Models in Human-Computer Interaction: Research Issues About What the User of Software Knows. John M. Carroll, Judith Reitman Olson, and Nancy Anderson. June 1987.

13. Extracting Expertise from Experts: Methods for Knowledge Acquisition. Judith Reitman Olson and Henry Rueter. June 1987.

14. An Advantage Model of Choice. Eldar B. Shafir, Daniel N. Osherson, and Edward E. Smith. September 1987.

15. Can Principles of Cognition Lower the Barriers to Programming? Clayton Lewis and Gary M. Olson. September 1987.

16. Management Use of Computer Technology: A Comparison of Two Theoretical Models. Richard P. Bagozzi, Fred D. Davis, and Paul R. Warshaw. March 1988.

17. Analysis of the Cognition Involved in Spreadsheet Software Interaction. Judith Reitman Olson, Erik Nilsen. April 1988.

18. Intellectual Development. Gary M. Olson. June 1988.

19. Induction and the Acquisition of English Auxiliaries: The Effects of Differentially Enriched Input. Marilyn Shatz, Erika Hoff-Ginsberg, Douglas MacIver. June 1988.

20. Modern Mental Chronometry. David E. Meyer, Allen M. Osman, David E. Irwin, Steven Yantis. October 1988.

21. Expertise and the Ability to Explain Audit Findings. Robert Libby, David M. Frederick. February 1989.

22. Supporting Collaboration with Advanced Multimedia Electronic Mail: The NSF EXPRES Project. Gary M. Olson, Daniel E. Atkins. February, 1989.

23. Category Based Induction. Daniel N. Osherson, Edward E. Smith, Ormond Wilkie, Alejandro Lopez, Eldar Shafir. March 1989.

24. Computer Aided Group Judgment: A Study of Group Decision Support Systems Effectiveness. Yue Kee Wong. August 1989.

25. CSCW: Evolution and Status of Computer Supported Cooperative Work. Paul Cornell, Robert Luchetti, Lisbeth Mack, Gary M. Olson. August 1989.

26. The Growth of Cognitive Modeling in Human Computer Interaction Since GOMS. Judith R. Olson, Gary M. Olson. November 1989.

27. Dimensional Overlap--Cognitive Basis for Stimulus-Response Compatibility: A Model and Taxonomy. Sylvan Korblum, Thierry Hasbroucq, Allen Osman. December 1989.

28. Probabilistic Forecasts of Stock Prices and Earnings: The Hazards of Nascent Expertise. J. Frank Yates, Linda S. McDaniel, Eric S. Brown. December 1989.

29. Heuristics of Reasoning and Analogy in Children's Visual Perspective Taking. Ilan Yaniv, Marilyn Shatz. February 1990.

30. Constraints on the Acquisition of English Modals. Marilyn Shatz, Sharon Wilcox. February 1990.

31. Patterns of Language Learning Related Behaviors: Evidence for Self-help in Acquiring Grammar. Marilyn Shatz, Karen Ebeling. February 1990.

32. User -Centered Design of Collaboration Technology. Gary M. Olson, Judith S. Olson. April 1990.

33. Designing Flexible Facilities for the Support of Collaboration. Gary M. Olson, Judith S. Olson, Lola J. Killey, Lisbeth A. Mack, Paul Cornell, Robert Luchetti. September 1990.

34. Techniques for Representing Expert Knowledge. Judith Reitman Olson, Kevin Biosi. November 1990.

35. Case for Rules in Reasoning. Edward E. Smith, Christopher Langston, Richard E. Nisbett. March 1991.